CS2030S
Recitation 2

Brian Cheong

Office hours

Office hours

- Fridays 4pm — 5pm

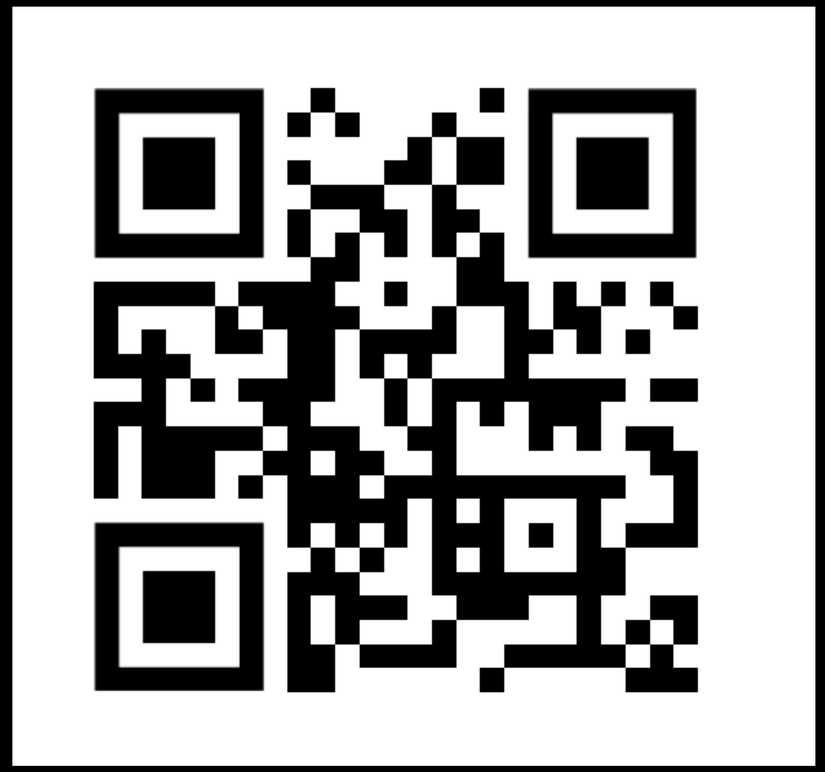
Office hours

- Fridays 4pm — 5pm
- Ask about creating telegroup
  - Can coordinate whether people coming, how many etc

Office hours

- Fridays 4pm — 5pm
- Ask about creating telegroup
  - Can coordinate whether people coming, how many etc
- Venue: COM2-B1-03
  - Walk past coolspot and go down staircase
  - There’s a dungeon there, that’s my office (near LT19)
  - labelled TA/GT cluster 3 and knock on the door

telegroup



A little bit side quest

A little bit side quest

- Compilers

A little bit side quest

- Compilers
  - A Program that converts another program from one language into another one

A little bit side quest

- Compilers
  - A Program that converts another program from one language into another one
  - Latex into pdf, C source code into Assembly, source to javascript etc

A little bit side quest

- Compilers
  - A Program that converts another program from one language into another one
  - Latex into pdf, C source code into Assembly, source to javascript etc
- Interpreter

A little bit side quest

- Compilers
  - A Program that converts another program from one language into another one
  - Latex into pdf, C source code into Assembly, source to javascript etc
- Interpreter
  - Takes in a program and executes it

A little bit side quest

- Compilers
  - A Program that converts another program from one language into another one
  - Latex into pdf, C source code into Assembly, source to javascript etc
- Interpreter
  - Takes in a program and executes it
  - Your browser (javascript interpreter), JVM, computer processor

Dynamic binding

Dynamic binding

- When writing a Java program there are 2 processes
  - Compilation `javac`
  - Running/Execution `java`

Dynamic binding

- When writing a Java program there are 2 processes
  - Compilation `javac`
  - Running/Execution `java`
- These 2 programs run separately, so “see” things differently
  - Compiler translates Java source code into JVM bytecode
  - JVM interprets (runs/executes) the JVM bytecode produced

Compile-time

Compile-time

- Compiler can only see **compile-time** type
  - When you declare a variable/field with a type
  - that type is the compile-time type

Compile-time

- Compiler can only see **compile-time** type
  - When you declare a variable/field with a type
  - that type is the compile-time type
- Compiler wants to have type safe code as much as possible
  - should not call methods that may not exist for that type (compile-type)
  - Only allows invocation of methods that type has

Runtime-type

Runtime-type

- The actual type of the object that is on the heap
    - Interpreter will know what is the actual type as it runs things (compiler doesn’t have enough info)

Example

Example

- `Animal a = new Dog();`

Example

- `Animal a = new Dog();`
- Here, CTT(a) = Animal, RTT(a) = Dog

Dynamic binding steps

Dynamic binding steps

- Assuming `b.foo(c)`

Dynamic binding steps

- Assuming `b.foo(c)`
- During compilation

Dynamic binding steps

- Assuming `b.foo(c)`
- During compilation
  - Find methods of CTT(b)

Dynamic binding steps

- Assuming `b.foo(c)`
- During compilation
  - Find methods of CTT(b)
  - Look for `foo` that accepts CTT(c) (CTT(c) is a subtype of the parameter)

Dynamic binding steps

- Assuming `b.foo(c)`
- During compilation
  - Find methods of CTT(b)
  - Look for `foo` that accepts CTT(c) (CTT(c) is a subtype of the parameter)
  - If multiple methods match, take the most specific parameter (the “smaller” one)

Dynamic binding steps

- Assuming `b.foo(c)`
- During compilation
  - Find methods of CTT(b)
  - Look for `foo` that accepts CTT(c) (CTT(c) is a subtype of the parameter)
  - If multiple methods match, take the most specific parameter (the “smaller” one)
  - that `foo` is now the method descriptor (name, return type and parameter types) that we have chosen [COMPILATION DONE]

Dynamic binding steps

- Assuming `b.foo(c)`
- During compilation
  - Find methods of CTT(b)
  - Look for `foo` that accepts CTT(c) (CTT(c) is a subtype of the parameter)
  - If multiple methods match, take the most specific parameter (the “smaller” one)
  - that `foo` is now the method descriptor (name, return type and parameter types) that we have chosen [COMPILATION DONE]
- Note how we don’t use runtime type at all

Dynamic binding steps

Dynamic binding steps

- During runtime

Dynamic binding steps

- During runtime
  - We have the method descriptor from before

Dynamic binding steps

- During runtime
  - We have the method descriptor from before
  - Look for that EXACT descriptor in class of RTT(b) and run that

Dynamic binding steps

- During runtime
  - We have the method descriptor from before
  - Look for that EXACT descriptor in class of RTT(b) and run that
  - If RTT(b) does not have have an implementation, look for it’s inherited implementation

Q3b

- o1.equals(o2);
- o1.equals((Circle) o2);
- o1.equals(c2);
- c1.equals(o2);
- c1.equals((Circle) o2);
- c1.equals(c2);

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(o2);

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(o2);
  - CTT(o1) = Object

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(o2);
  - CTT(o1) = Object
  - CTT(o2) = Object

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(o2);
  - CTT(o1) = Object
  - CTT(o2) = Object
  - Methods available = boolean equals(Object)

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(o2);
  - CTT(o1) = Object
  - CTT(o2) = Object
  - Methods available = boolean equals(Object)
  - CTT(o2) <: Object so it’s ok

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(o2);
  - CTT(o1) = Object
  - CTT(o2) = Object
  - Methods available = boolean equals(Object)
  - CTT(o2) <: Object so it’s ok
  - Method descriptor boolean equals(Object)

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(o2);
  - CTT(o1) = Object
  - CTT(o2) = Object
  - Methods available = boolean equals(Object)
  - CTT(o2) <: Object so it's ok
  - Method descriptor boolean equals(Object)
  - Implementation found in circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(o2);
  - CTT(o1) = Object
  - CTT(o2) = Object
  - Methods available = boolean equals(Object)
  - CTT(o2) <: Object so it's ok
  - Method descriptor boolean equals(Object)
  - Implementation found in circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals((Circle) o2);

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals((Circle) o2);
  - CTT(o1) = Object

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals((Circle) o2);
  - CTT(o1) = Object
  - CTT((Circle) o2) = Circle

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals((Circle) o2);
  - CTT(o1) = Object
  - CTT((Circle) o2) = Circle
  - Methods available = boolean equals(Object)

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals((Circle) o2);
  - CTT(o1) = Object
  - CTT((Circle) o2) = Circle
  - Methods available = boolean equals(Object)
  - CTT((Circle) o2) <: Object so it’s ok

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals((Circle) o2);
  - CTT(o1) = Object
  - CTT((Circle) o2) = Circle
  - Methods available = boolean equals(Object)
  - CTT((Circle) o2) <: Object so it’s ok
  - Method descriptor boolean equals(Object)

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals((Circle) o2);
  - CTT(o1) = Object
  - CTT((Circle) o2) = Circle
  - Methods available = boolean equals(Object)
  - CTT((Circle) o2) <: Object so it’s ok
  - Method descriptor boolean equals(Object)
  - Implementation found in circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals((Circle) o2);
  - CTT(o1) = Object
  - CTT((Circle) o2) = Circle
  - Methods available = boolean equals(Object)
  - CTT((Circle) o2) <: Object so it’s ok
  - Method descriptor boolean equals(Object)
  - Implementation found in circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(c2);

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(c2);
  - CTT(o1) = Object

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(c2);
  - CTT(o1) = Object
  - CTT(c2) = Circle

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(c2);
  - CTT(o1) = Object
  - CTT(c2) = Circle
  - Methods available = boolean equals(Object)

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(c2);
  - CTT(o1) = Object
  - CTT(c2) = Circle
  - Methods available = boolean equals(Object)
  - CTT(c2) <: Object so it’s ok

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(c2);
  - CTT(o1) = Object
  - CTT(c2) = Circle
  - Methods available = boolean equals(Object)
  - CTT(c2) <: Object so it’s ok
  - Method descriptor boolean equals(Object)

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(c2);
  - CTT(o1) = Object
  - CTT(c2) = Circle
  - Methods available = boolean equals(Object)
  - CTT(c2) <: Object so it’s ok
  - Method descriptor boolean equals(Object)
  - Implementation found in circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- o1.equals(c2);
  - CTT(o1) = Object
  - CTT(c2) = Circle
  - Methods available = boolean equals(Object)
  - CTT(c2) <: Object so it's ok
  - Method descriptor boolean equals(Object)
  - Implementation found in circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(o2);

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(o2);
  - CTT(c1) = Circle

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(o2);
  - CTT(c1) = Circle
  - CTT(o2) = Object

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(o2);
  - CTT(c1) = Circle
  - CTT(o2) = Object
  - Methods available = boolean equals(Object) and boolean equals(Circle)

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(o2);
  - CTT(c1) = Circle
  - CTT(o2) = Object
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object
    but CTT(o2) </: Circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(o2);
  - CTT(c1) = Circle
  - CTT(o2) = Object
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object
    but CTT(o2) </: Circle
  - Method descriptor boolean equals(Object)

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(o2);
  - CTT(c1) = Circle
  - CTT(o2) = Object
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object
    but CTT(o2) </: Circle
  - Method descriptor boolean equals(Object)
  - Implementation found in circle

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(o2);
  - CTT(c1) = Circle
  - CTT(o2) = Object
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object
    but CTT(o2) </: Circle
  - Method descriptor boolean equals(Object)
  - Implementation found in circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals((Circle) o2);

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals((Circle) o2);
  - CTT(c1) = Circle

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals((Circle) o2);
  - CTT(c1) = Circle
  - CTT((Circle) o2) = Circle

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals((Circle) o2);
  - CTT(c1) = Circle
  - CTT((Circle) o2) = Circle
  - Methods available = boolean equals(Object) and boolean equals(Circle)

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals((Circle) o2);
  - CTT(c1) = Circle
  - CTT((Circle) o2) = Circle
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object
    and CTT((Circle) o2) <: Circle
    take more specific

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals((Circle) o2);
  - CTT(c1) = Circle
  - CTT((Circle) o2) = Circle
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object
    and CTT((Circle) o2) <: Circle
    take more specific
  - Method descriptor boolean equals(Circle)

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals((Circle) o2);
  - CTT(c1) = Circle
  - CTT((Circle) o2) = Circle
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object
    and CTT((Circle) o2) <: Circle
    take more specific
  - Method descriptor boolean equals(Circle)
  - Implementation found in circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals((Circle) o2);
  - CTT(c1) = Circle
  - CTT(o2) = Circle
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object
    and CTT((Circle) o2) <: Circle
    take more specific
  - Method descriptor boolean equals(Circle)
  - Implementation found in circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(c2);

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(c2);
  - CTT(c1) = Circle

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(c2);
  - CTT(c1) = Circle
  - CTT(c2) = Circle

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(c2);
  - CTT(c1) = Circle
  - CTT(c2) = Circle
  - Methods available = boolean equals(Object) and boolean equals(Circle)

```java
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(c2);
  - CTT(c1) = Circle
  - CTT(c2) = Circle
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object and CTT(c2) <: Circle take more specific

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(c2);
  - CTT(c1) = Circle
  - CTT(c2) = Circle
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object
    and CTT(c2) <: Circle
    take more specific
  - Method descriptor boolean equals(Circle)

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(c2);
  - CTT(c1) = Circle
  - CTT(c2) = Circle
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object
    and CTT(c2) <: Circle
    take more specific
  - Method descriptor boolean equals(Circle)
  - Implementation found in circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Q3b

- c1.equals(c2);
  - CTT(c1) = Circle
  - CTT(c2) = Circle
  - Methods available = boolean equals(Object) and boolean equals(Circle)
  - CTT(o2) <: Object
    and CTT(c2) <: Circle
    take more specific
  - Method descriptor boolean equals(Circle)
  - Implementation found in circle

```
public class Circle {
  private Point centre;
  private int radius;

  public Circle(Point centre, int radius) {
    this.centre = centre;
    this.radius = radius;
  }

  @Override
  public boolean equals(Object obj) {
    System.out.println("equals(Object) called");
    if (obj == this) {
      return true;
    }
    if (obj instanceof Circle) {
      Circle circle = (Circle) obj;
      return (circle.centre.equals(centre) && circle.radius == radius);
    } else {
      return false;
    }
  }

  public boolean equals(Circle circle) {
    System.out.println("equals(Circle) called");
    return circle.centre.equals(centre) && circle.radius == radius;
  }
}
```

Problem Set 2

Liskov Substitution Principle

Liskov Substitution Principle

- Let $\phi(x)$ be a property provable about objects x of type T. Then $\phi(y)$ should be true for objects of type S where $S <: T$

Liskov Substitution Principle

- Let $\phi(x)$ be a property provable about objects x of type T. Then $\phi(y)$ should be true for objects of type S where $S <: T$
    - In english, if you have some property (user-defined) for a class/type

Liskov Substitution Principle

- Let $\phi(x)$ be a property provable about objects x of type T. Then $\phi(y)$ should be true for objects of type S where $S <: T$
  - In english, if you have some property (user-defined) for a class/type
  - It’s subtypes should maintain that property

Example

Example

- Notes example (Restaurant)

Example

- Notes example (Restaurant)
  - Every restaurant should be open from 12pm to 10pm

Example

- Notes example (Restaurant)
  - Every restaurant should be open from 12pm to 10pm
  - lunch one violates since it closes at 2pm

Example

- Notes example (Restaurant)
  - Every restaurant should be open from 12pm to 10pm
  - lunch one violates since it closes at 2pm
  - 24hr one is ok

Example

- Notes example (Restaurant)
  - Every restaurant should be open from 12pm to 10pm
  - lunch one violates since it closes at 2pm
  - 24hr one is ok
  - (is restaurant) ^ (12pm <= time <= 10pm) —> Open

Example

- Notes example (Restaurant)
  - Every restaurant should be open from 12pm to 10pm
  - lunch one violates since it closes at 2pm
  - 24hr one is ok
  - (is restaurant) ^ (12pm <= time <= 10pm) —> Open
  - if predicate is false, then anything can happen (doesn’t have to be closed or open)

Q1a

Q1a

- `Rectangle` class
  - `Rectangle::getArea` is expected to return product of height and width

Q1a

- `Rectangle` class
  - `Rectangle::getArea` is expected to return product of height and width
- Design a class `Square` that inherits from `Rectangle`
  - New constraint: all 4 sides are always of the same length

Q1a

- `Rectangle` class
  - `Rectangle::getArea` is expected to return product of height and width
- Design a class `Square` that inherits from `Rectangle`
  - New constraint: all 4 sides are always of the same length
- Create class `Square` with a single constructor method

Q1a

```
class Square {

}
```

```
public class Rectangle {
  private double width;
  private double height;

  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }

  public double getArea() {
    return this.width * this.height;
  }

  @Override
  public String toString() {
    return "Width: " + this.width + " Height: " +
this.height;
  }
}
```

Q1a

```
class Square extends Rectangle {

}
```

```
public class Rectangle {
  private double width;
  private double height;

  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }

  public double getArea() {
    return this.width * this.height;
  }

  @Override
  public String toString() {
    return "Width: " + this.width + " Height: " +
this.height;
  }
}
```

Q1a

```
class Square extends Rectangle {
  public Square(double length) {

  }
}
```

```
public class Rectangle {
  private double width;
  private double height;

  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }

  public double getArea() {
    return this.width * this.height;
  }

  @Override
  public String toString() {
    return "Width: " + this.width + " Height: " +
this.height;
  }
}
```

Q1a

```
class Square extends Rectangle {
  public Square(double length) {
    super(length, length);
  }
}
```

```
public class Rectangle {
  private double width;
  private double height;

  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }

  public double getArea() {
    return this.width * this.height;
  }

  @Override
  public String toString() {
    return "Width: " + this.width + " Height: " +
this.height;
  }
}
```

Q1b

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

Q1b

- We have 2 setters for height and width within Rectangle

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

Q1b

- We have 2 setters for height and width within Rectangle
- Property:

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

Q1b

- We have 2 setters for height and width within Rectangle
- Property:
  - w = last width set

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

Q1b

- We have 2 setters for height and width within Rectangle
- Property:
  - w = last width set
  - h = last height set

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

Q1b

- We have 2 setters for height and width within Rectangle
- Property:
  - w = last width set
  - h = last height set
  - then `getArea( )` must return w * h

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

Q1b

- We have 2 setters for height and width within Rectangle
- Property:
  - w = last width set
  - h = last height set
  - then `getArea()` must return w * h
- Why is this undesirable for Square?

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

Q1b

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

Q1b

- Why is this undesirable for Square?

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

Q1b

- Why is this undesirable for Square?
  - Now we can have set the height and width independently for Square

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

Q1b

- Why is this undesirable for Square?
  - Now we can have set the height and width independently for Square
  - Violates the other property that Square must have 4 sides of equal length

```
public void setHeight(double height) {
  this.height = height;
}

public void setWidth(double width) {
  this.width = width;
}
```

Q1c

```
@Override
public void setHeight(double height) {
  super.setHeight(height);
  super.setWidth(height);
}

@Override
public void setWidth(double width) {
  super.setHeight(width);
  super.setWidth(width);
}
```

Q1c

- By LSP, anywhere that expects Rectangle, we can put a Square

```
@Override
public void setHeight(double height) {
  super.setHeight(height);
  super.setWidth(height);
}

@Override
public void setWidth(double width) {
  super.setHeight(width);
  super.setWidth(width);
}
```

Q1c

- By LSP, anywhere that expects Rectangle, we can put a Square
- Imagine a method that takes in a Rectangle
  - It will expect properties of rectangle

```
@Override
public void setHeight(double height) {
  super.setHeight(height);
  super.setWidth(height);
}

@Override
public void setWidth(double width) {
  super.setHeight(width);
  super.setWidth(width);
}
```

Q1c

- By LSP, anywhere that expects Rectangle, we can put a Square
- Imagine a method that takes in a Rectangle
  - It will expect properties of rectangle
- If we pass in Square this no longer works

```
@Override
public void setHeight(double height) {
  super.setHeight(height);
  super.setWidth(height);
}

@Override
public void setWidth(double width) {
  super.setHeight(width);
  super.setWidth(width);
}
```

Q1c

- By LSP, anywhere that expects Rectangle, we can put a Square
- Imagine a method that takes in a Rectangle
  - It will expect properties of rectangle
- If we pass in Square this no longer works
- We fixed 4 sides being same but violated other property

```
@Override
public void setHeight(double height) {
  super.setHeight(height);
  super.setWidth(height);
}

@Override
public void setWidth(double width) {
  super.setHeight(width);
  super.setWidth(width);
}
```

Q1d

- So then does it make sense for Rectangle to inherit from Square

Q1d

- So then does it make sense for Rectangle to inherit from Square
  - No

Q1d

- So then does it make sense for Rectangle to inherit from Square
  - No
  - Square has a constraint that all 4 sides must be of equal length

Q1d

- So then does it make sense for Rectangle to inherit from Square
  - No
  - Square has a constraint that all 4 sides must be of equal length
  - Rectangle relaxes this constraint

Q1d

- So then does it make sense for Rectangle to inherit from Square
  - No
  - Square has a constraint that all 4 sides must be of equal length
  - Rectangle relaxes this constraint
  - Better for Square and Rectangle to not inherit from each other at all

Q2a

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2a

- Which statements are allowed?

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2a

- Which statements are allowed?
  - `s.print();`

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2a

- Which statements are allowed?
  - `s.print();`
  - `p.print();`

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2a

- Which statements are allowed?
  - `s.print();`
  - `p.print();`
  - `s.getArea();`

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2a

- Which statements are allowed?
  - `s.print();`
  - `p.print();`
  - `s.getArea();`
  - `p.getArea();`

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2a

- Which statements are allowed?
  - `s.print();`
  - `p.print();`
  - `s.getArea();`
  - `p.getArea();`
- Why does the compiler not allow some of these statements?

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2a

- Which statements are allowed?
  - `s.print();`
  - `p.print();`
  - `s.getArea();`
  - `p.getArea();`
- Why does the compiler not allow some of these statements?
  - Preserve type safety

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2a

- Which statements are allowed?
  - `s.print();`
  - `p.print();`
  - `s.getArea();`
  - `p.getArea();`
- Why does the compiler not allow some of these statements?
  - Preserve type safety
  - Compiler doesn’t know RTT

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2a

- Which statements are allowed?
  - `s.print();`
  - `p.print();`
  - `s.getArea();`
  - `p.getArea();`
- Why does the compiler not allow some of these statements?
  - Preserve type safety
  - Compiler doesn’t know RTT
  - Only knows about the CTT methods

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2a

- Which statements are allowed?
  - `s.print();`
  - `p.print();`
  - `s.getArea();`
  - `p.getArea();`
- Why does the compiler not allow some of these statements?
  - Preserve type safety
  - Compiler doesn’t know RTT
  - Only knows about the CTT methods
  - Only allow methods that is guaranteed to exist for that compile type

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2b

- Ok then can we implement as an abstract class then?

```
public abstact class Shape {
  public double getArea();
}

public abstract class Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2b

- Ok then can we implement as an abstract class then?
  - No, java doesn’t allow classes to inherit from multiple classes

```
public abstact class Shape {
  public double getArea();
}

public abstract class Printable {
  public void print();
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2c

- Ok then can we have another interface that extends both Shape and Printable?

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}

public interface PrintableShape
    extends Printable, Shape {
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

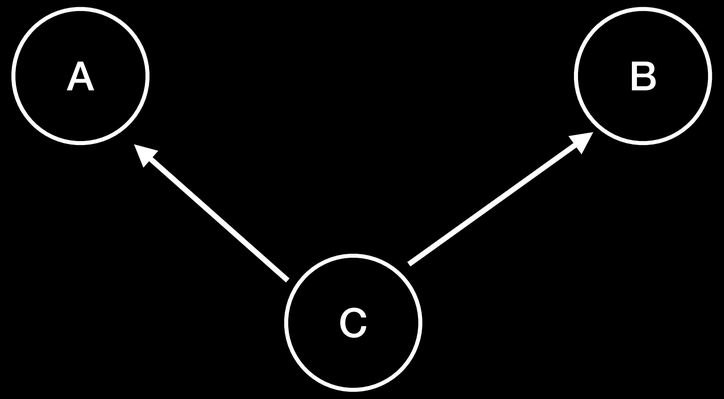
Q2c

- Ok then can we have another interface that extends both Shape and Printable?
  - Yeah

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}

public interface PrintableShape
    extends Printable, Shape {
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2d

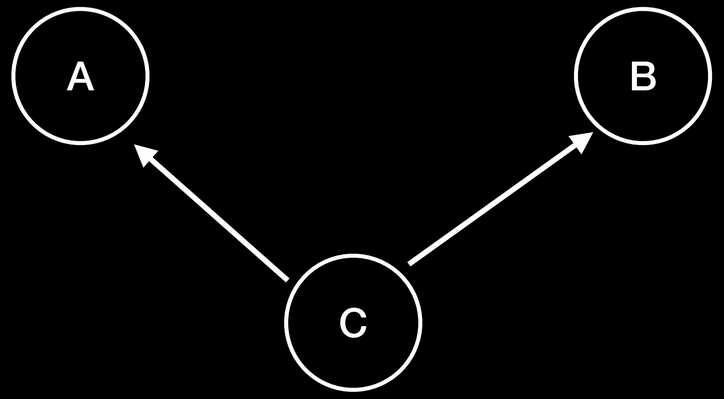



```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}

public interface PrintableShape
    extends Printable, Shape {
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2d

- Ok why multiple interface can but multiple classes cannot?

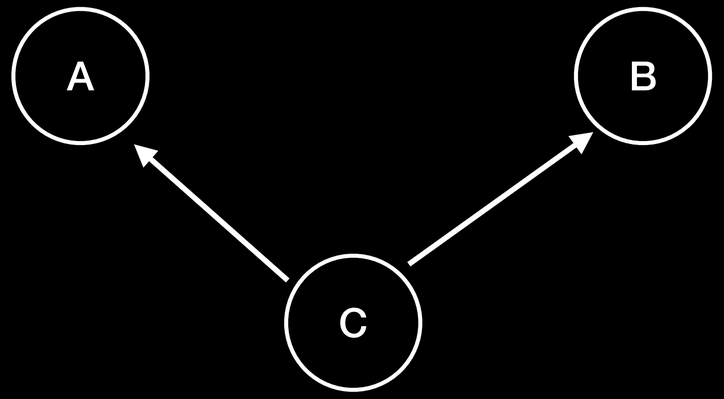



```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}

public interface PrintableShape
    extends Printable, Shape {
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2d

- Ok why multiple interface can but multiple classes cannot?

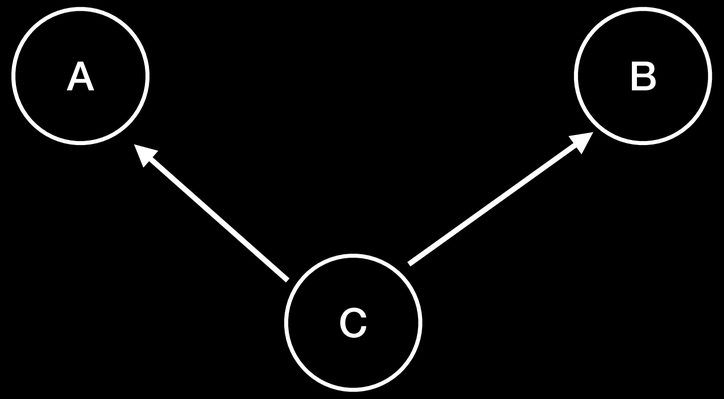



```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}

public interface PrintableShape
    extends Printable, Shape {
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2d

- Ok why multiple interface can but multiple classes cannot?

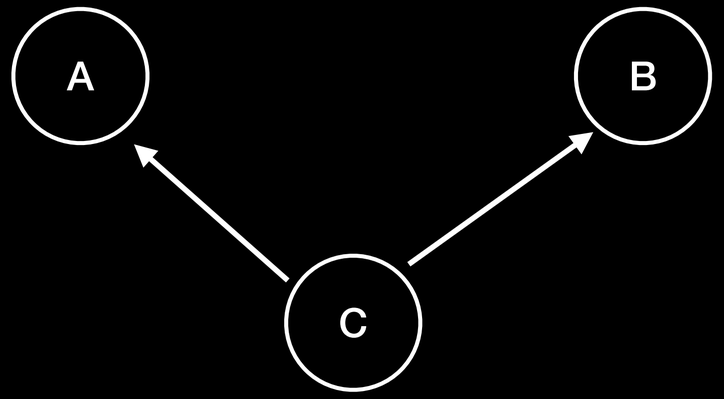



```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}

public interface PrintableShape
    extends Printable, Shape {
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2d

- Ok why multiple interface can but multiple classes cannot?

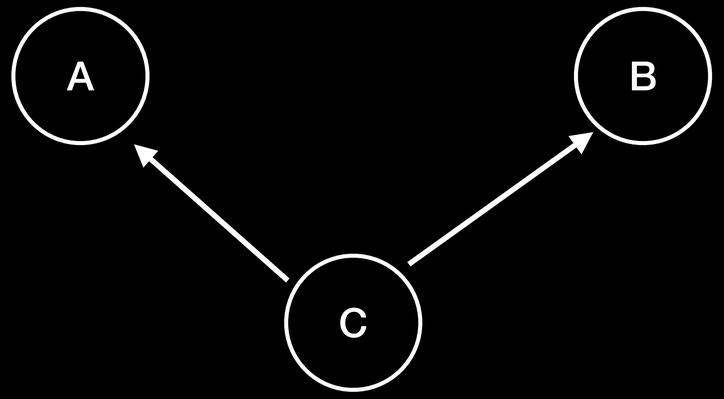



```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}

public interface PrintableShape
    extends Printable, Shape {
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2d

- Ok why multiple interface can but multiple classes cannot?



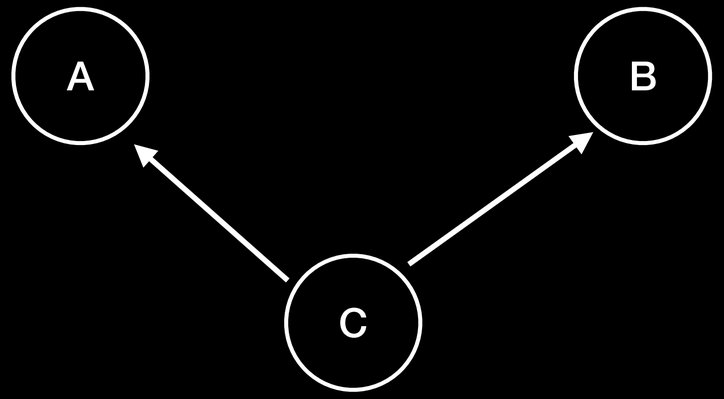
- What if A and B both have an implementation of some method foo

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}

public interface PrintableShape
    extends Printable, Shape {
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2d

- Ok why multiple interface can but multiple classes cannot?



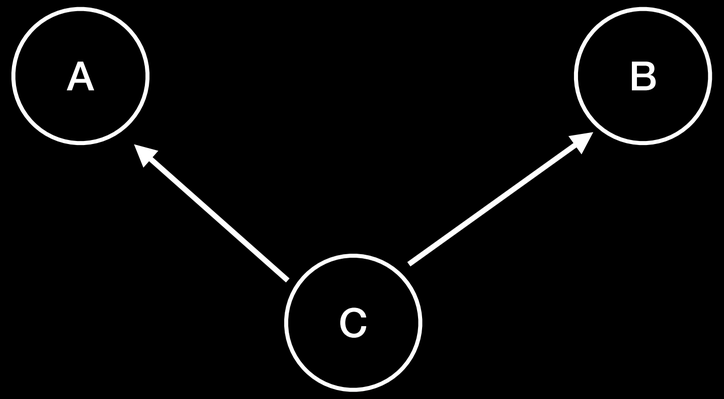
- What if A and B both have an implementation of some method foo
- Which does C inherit?

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}

public interface PrintableShape
    extends Printable, Shape {
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Q2d

- Ok why multiple interface can but multiple classes cannot?



- What if A and B both have an implementation of some method foo
- Which does C inherit?
- If interface there is no implementation so its fine, a concrete class has to implement

```
public interface Shape {
  public double getArea();
}

public interface Printable {
  public void print();
}

public interface PrintableShape
    extends Printable, Shape {
}
```

```
Circle c = new Circle(new Point(0,0), 10);
Shape s = c;
Printable p = c;
```

Thank you see you next week :)