CS2030S Recitation

Problem Set 1

Brian Cheong

About me

About me

- Brian Cheong

About me

- Brian Cheong
- Studied CS in NUS for UG

About me

- Brian Cheong
- Studied CS in NUS for UG
- Also a year 1

About me

- Brian Cheong
- Studied CS in NUS for UG
- Also a year 1
- Previously a CS2030S Lab TA for 3 years

About me

- Brian Cheong
- Studied CS in NUS for UG
- Also a year 1
- Previously a CS2030S Lab TA for 3 years
- Email: bskch@nus.edu.sg

Flow of recitation

Flow of recitation

- Please watch lecture before coming
  - Better use of time to watch lecture than to come for recitation (trust me)

Flow of recitation

- Please watch lecture before coming
  - Better use of time to watch lecture than to come for recitation (trust me)
- At least read through the questions

Flow of recitation

- Please watch lecture before coming
  - Better use of time to watch lecture than to come for recitation (trust me)
- At least read through the questions
- Let’s make this more interactive and discussion based

Flow of recitation

- Please watch lecture before coming
  - Better use of time to watch lecture than to come for recitation (trust me)
- At least read through the questions
- Let’s make this more interactive and discussion based
- If anything is unclear please stop me and ask
  - or look super confused

Promises from me

Promises from me

- I'll try to give you an intuitive understanding of the material
  - Sometimes lectures may be too much info
  - Try to simplify to make it easier to understand

Promises from me

- I'll try to give you an intuitive understanding of the material
  - Sometimes lectures may be too much info
  - Try to simplify to make it easier to understand
- Give you intuitions on how to approach questions

Promises from me

- I'll try to give you an intuitive understanding of the material
  - Sometimes lectures may be too much info
  - Try to simplify to make it easier to understand
- Give you intuitions on how to approach questions
- Will hold consultations (tbd) come if you need help

Some getting to know you questions

Some getting to know you questions

- How many prior experience with statically typed languages?

Some getting to know you questions

- How many prior experience with statically typed languages?
- How many prior experience with Java?

Q1a

We have the following java program

```
class BankAccount {
  double balance;

  BankAccount(double initBalance) {
    this.balance = initBalance;
  }
}

class Customer {
  BankAccount account;

  Customer() {
    this.account = new BankAccount(0);
  }

  public void deposit(double amount) {
    this.account.balance += amount;
  }
  public boolean withdraw(double amount) {
    if (this.account.balance ⩾ amount) {
      this.account.balance -= amount;
      return true;
```

Does this program follow the principle of information hiding? Explain.

Q1a

We have the following java program

```
1   class BankAccount {
2     double balance;
3
4     BankAccount(double initBalance) {
5       this.balance = initBalance;
6     }
7   }
8
9   class Customer {
10    BankAccount account;
11
12    Customer() {
13      this.account = new BankAccount(0);
14    }
15
16    public void deposit(double amount) {
17      this.account.balance += amount;
18    }
19    public boolean withdraw(double amount) {
20      if (this.account.balance ⩾ amount) {
21        this.account.balance -= amount;
22        return true;
```

Does this program follow the principle of information hiding? Explain.

- No

Q1a

We have the following java program

```
1   class BankAccount {
2     double balance;
3
4     BankAccount(double initBalance) {
5       this.balance = initBalance;
6     }
7   }
8
9   class Customer {
10    BankAccount account;
11
12    Customer() {
13      this.account = new BankAccount(0);
14    }
15
16    public void deposit(double amount) {
17      this.account.balance += amount;
18    }
19    public boolean withdraw(double amount) {
20      if (this.account.balance ⩾ amount) {
21        this.account.balance -= amount;
22        return true;
```

Does this program follow the principle of information hiding? Explain.

- No
  - `balance` in `BankAccount` is publically accessible

Q1a

We have the following java program

```
1   class BankAccount {
2     double balance;
3
4     BankAccount(double initBalance) {
5       this.balance = initBalance;
6     }
7   }
8
9   class Customer {
10    BankAccount account;
11
12    Customer() {
13      this.account = new BankAccount(0);
14    }
15
16    public void deposit(double amount) {
17      this.account.balance += amount;
18    }
19    public boolean withdraw(double amount) {
20      if (this.account.balance ⩾ amount) {
21        this.account.balance -= amount;
22        return true;
```

Does this program follow the principle of information hiding? Explain.

- No
  - `balance` in `BankAccount` is publically accessible
  - `account` in `Customer` is publically accessible too

Q1a

We have the following java program

```
1   class BankAccount {
2     double balance;
3
4     BankAccount(double initBalance) {
5       this.balance = initBalance;
6     }
7   }
8
9   class Customer {
10    BankAccount account;
11
12    Customer() {
13      this.account = new BankAccount(0);
14    }
15
16    public void deposit(double amount) {
17      this.account.balance += amount;
18    }
19    public boolean withdraw(double amount) {
20      if (this.account.balance ⩾ amount) {
21        this.account.balance -= amount;
22        return true;
```

Does this program follow the principle of information hiding? Explain.

- No
  - `balance` in `BankAccount` is publically accessible
  - `account` in `Customer` is publically accessible too
- Always try to keep fields private

Q1b

Does this program follow the principle of "Tell, Don't Ask"? Explain.

We have the following java program

```
1   class BankAccount {
2     double balance;
3
4     BankAccount(double initBalance) {
5       this.balance = initBalance;
6     }
7   }
8
9   class Customer {
10    BankAccount account;
11
12    Customer() {
13      this.account = new BankAccount(0);
14    }
15
16    public void deposit(double amount) {
17      this.account.balance += amount;
18    }
19    public boolean withdraw(double amount) {
20      if (this.account.balance ⩾ amount) {
21        this.account.balance -= amount;
22        return true;
```

Q1b

We have the following java program

```
1   class BankAccount {
2     double balance;
3
4     BankAccount(double initBalance) {
5       this.balance = initBalance;
6     }
7   }
8
9   class Customer {
10    BankAccount account;
11
12    Customer() {
13      this.account = new BankAccount(0);
14    }
15
16    public void deposit(double amount) {
17      this.account.balance += amount;
18    }
19    public boolean withdraw(double amount) {
20      if (this.account.balance ⩾ amount) {
21        this.account.balance -= amount;
22        return true;
```

Does this program follow the principle of "Tell, Don't Ask"? Explain.

- No

Q1b

We have the following java program

```
1   class BankAccount {
2     double balance;
3
4     BankAccount(double initBalance) {
5       this.balance = initBalance;
6     }
7   }
8
9   class Customer {
10    BankAccount account;
11
12    Customer() {
13      this.account = new BankAccount(0);
14    }
15
16    public void deposit(double amount) {
17      this.account.balance += amount;
18    }
19    public boolean withdraw(double amount) {
20      if (this.account.balance ⩾ amount) {
21        this.account.balance -= amount;
22        return true;
```

Does this program follow the principle of "Tell, Don't Ask"? Explain.

- No
  - `Customer` asks for the balance from `BankAccount` in the `withdraw` method and does the computation

Q1b

We have the following java program

```
1  class BankAccount {
2    double balance;
3
4    BankAccount(double initBalance) {
5      this.balance = initBalance;
6    }
7  }
8
9  class Customer {
10   BankAccount account;
11
12   Customer() {
13     this.account = new BankAccount(0);
14   }
15
16   public void deposit(double amount) {
17     this.account.balance += amount;
18   }
19   public boolean withdraw(double amount) {
20     if (this.account.balance ⩾ amount) {
21       this.account.balance -= amount;
22       return true;
```

Does this program follow the principle of "Tell, Don't Ask"? Explain.

- No
  - `Customer` asks for the balance from `BankAccount` in the `withdraw` method and does the computation
  - `Customer` should **tell** `BankAccount` to do the withdraw internally

Q1b

We have the following java program

```
1   class BankAccount {
2     double balance;
3
4     BankAccount(double initBalance) {
5       this.balance = initBalance;
6     }
7   }
8
9   class Customer {
10    BankAccount account;
11
12    Customer() {
13      this.account = new BankAccount(0);
14    }
15
16    public void deposit(double amount) {
17      this.account.balance += amount;
18    }
19    public boolean withdraw(double amount) {
20      if (this.account.balance ⩾ amount) {
21        this.account.balance -= amount;
22        return true;
```

Does this program follow the principle of "Tell, Don't Ask"? Explain.

- No
  - `Customer` asks for the balance from `BankAccount` in the `withdraw` method and does the computation
  - `Customer` should **tell** `BankAccount` to do the withdraw internally
  - similar situation for `deposit`

Q1b

We have the following java program

```
1   class BankAccount {
2     double balance;
3
4     BankAccount(double initBalance) {
5       this.balance = initBalance;
6     }
7   }
8
9   class Customer {
10    BankAccount account;
11
12    Customer() {
13      this.account = new BankAccount(0);
14    }
15
16    public void deposit(double amount) {
17      this.account.balance += amount;
18    }
19    public boolean withdraw(double amount) {
20      if (this.account.balance ⩾ amount) {
21        this.account.balance -= amount;
22        return true;
```

Does this program follow the principle of "Tell, Don't Ask"? Explain.

- No
  - `Customer` asks for the balance from `BankAccount` in the `withdraw` method and does the computation
  - `Customer` should **tell** `BankAccount` to do the withdraw internally
  - similar situation for `deposit`
- Don't get internals of your fields and do computations for it

Q1b

We have the following java program

```java
class BankAccount {
  double balance;

  BankAccount(double initBalance) {
    this.balance = initBalance;
  }
}

class Customer {
  BankAccount account;

  Customer() {
    this.account = new BankAccount(0);
  }

  public void deposit(double amount) {
    this.account.balance += amount;
  }
  public boolean withdraw(double amount) {
    if (this.account.balance ⩾ amount) {
      this.account.balance -= amount;
      return true;
```

Does this program follow the principle of "Tell, Don't Ask"? Explain.

- No
  - `Customer` asks for the balance from `BankAccount` in the `withdraw` method and does the computation
  - `Customer` should **tell** `BankAccount` to do the withdraw internally
  - similar situation for `deposit`
- Don't get internals of your fields and do computations for it
- Try to push the computation within the class of the respective field

Q1b

We have the following java program

```
1  class BankAccount {
2    double balance;
3
4    BankAccount(double initBalance) {
5      this.balance = initBalance;
6    }
7  }
8
9  class Customer {
10   BankAccount account;
11
12   Customer() {
13     this.account = new BankAccount(0);
14   }
15
16   public void deposit(double amount) {
17     this.account.balance += amount;
18   }
19   public boolean withdraw(double amount) {
20     if (this.account.balance ⩾ amount) {
21       this.account.balance -= amount;
22       return true;
```

Does this program follow the principle of "Tell, Don't Ask"? Explain.

- No
  - `Customer` asks for the balance from `BankAccount` in the `withdraw` method and does the computation
  - `Customer` should **tell** `BankAccount` to do the withdraw internally
  - similar situation for `deposit`
- Don't get internals of your fields and do computations for it
- Try to push the computation within the class of the respective field
  - Pushing `withdraw` computation within `BankAccount`

Q1c

Fix the code

```
class BankAccount {
  double balance;

  BankAccount(double initBalance) {
    this.balance = initBalance;
  }
}

class Customer {
  BankAccount account;

  Customer() {
    this.account = new BankAccount(0);
  }

  public void deposit(double amount) {
    this.account.balance += amount;
  }
  public boolean withdraw(double amount) {
    if (this.account.balance ≥ amount) {
      this.account.balance -= amount;
      return true;
```

Q1c

Fix the code

```
class BankAccount {
  private double balance;

  BankAccount(double initBalance) {
    this.balance = initBalance;
  }
}

class Customer {
  private BankAccount account;

  Customer() {
    this.account = new BankAccount(0);
  }

  public void deposit(double amount) {
    this.account.balance += amount;
  }
  public boolean withdraw(double amount) {
    if (this.account.balance >= amount) {
      this.account.balance -= amount;
      return true;
```

Q1c

Fix the code

```
class BankAccount {
  private double balance;

  BankAccount(double initBalance) {
    this.balance = initBalance;
  }

  public void deposit(double amount) {
    this.balance += amount;
  }

  public boolean withdraw(double amount) {
    if (this.balance ≥ amount) {
      this.balance -= amount;
      return true;
    }
    return false;
  }
}

class Customer {
  private BankAccount account;
```

Q1c

Fix the code

```
class BankAccount {
  private double balance;

  BankAccount(double initBalance) {
    this.balance = initBalance;
  }

  public void deposit(double amount) {
    this.balance += amount;
  }

  public boolean withdraw(double amount) {
    if (this.balance ≥ amount) {
      this.balance -= amount;
      return true;
    }
    return false;
  }
}

class Customer {
  private BankAccount account;
```

Q2a

Consider the following code

```
1   class Vector2D {
2     private double x;
3     private double y;
4
5     public Vector2D(double x, double y) {
6       this.x = x;
7       this.y = y;
8     }
9
10    public void add(Vector2D v) {
11      this.x = this.x + v.x;
12      this.y = this.y + v.y;
13      // line A
14    }
15  }
```

Stack

Heap

Stack

- Made of frames

Heap

Stack

- Made of frames
  - The bindings between variable names and its value

Heap

Stack

- Made of frames
  - The bindings between variable names and its value
- 1 call 1 frame created

Heap

Stack

- Made of frames
  - The bindings between variable names and its value
- 1 call 1 frame created
- Call finishes? Frame is removed

Heap

Stack

- Made of frames
  - The bindings between variable names and its value
- 1 call 1 frame created
- Call finishes? Frame is removed

Heap

- Where objects that are created live

Stack

- Made of frames
  - The bindings between variable names and its value
- 1 call 1 frame created
- Call finishes? Frame is removed

Heap

- Where objects that are created live
- Objects contain information of that instance
  - mainly fields (more to come)

Stack

- Made of frames
  - The bindings between variable names and its value
- 1 call 1 frame created
- Call finishes? Frame is removed

Heap

- Where objects that are created live
- Objects contain information of that instance
  - mainly fields (more to come)
- Why need the heap? why not everything in the stack?
  - Objects can "live" on after stack frame removed

Q2a

Now we have the following statements in the `main` method. What does the stack and heap diagram look like?

```
Vector2D v1 = new Vector2D(1, 1);
Vector2D v2 = new Vector2D(2, 2);
v1.add(v2);
```

```
1  class Vector2D {
2    private double x;
3    private double y;
4
5    public Vector2D(double x, double y) {
6      this.x = x;
7      this.y = y;
8    }
9
10   public void add(Vector2D v) {
11     this.x = this.x + v.x;
12     this.y = this.y + v.y;
13     // line A
14   }
15 }
```

Q2a

Stack

1 call, 1 frame

Stack if FILO

Q2a

Stack

1 call, 1 frame

- main method

Stack if FILO

Q2a

Stack

1 call, 1 frame

- main method
- add method

Stack if FILO

Q2a

Stack

1 call, 1 frame

- main method
- add method

Stack if FILO

- Grows upwards

Q2a

Stack

1 call, 1 frame

- main method
- add method

Stack if FILO

- Grows upwards
- which is first? main? add?

Q2a

Stack

1 call, 1 frame

- main method
- add method

Stack if FILO

- Grows upwards
- which is first? main? add?
  - main then add

Q2a

Stack

1 call, 1 frame

- main method
- add method

Stack if FILO

- Grows upwards
- which is first? main? add?
  - main then add
- What are the bindings in each frame?

Q2a

Stack

1 call, 1 frame

- main method
- add method

Stack if FILO

- Grows upwards
- which is first? main? add?
  - main then add
- What are the bindings in each frame?
  - what variables are there

Q2a

Stack

Heap

What objects are created?

Q2a

Stack

- For each variable what is the value?

Heap

What objects are created?

Q2a

Stack

- For each variable what is the value?
- Is the value a primitive(int, bool, etc)?

Heap

What objects are created?

Q2a

Stack

- For each variable what is the value?
- Is the value a primitive(int, bool, etc)?
  - Just put it in the box

Heap

What objects are created?

Q2a

Stack

- For each variable what is the value?
- Is the value a primitive(int, bool, etc)?
  - Just put it in the box
- Is it referring (pointing) to an object?

Heap

What objects are created?

Q2a

Stack

- For each variable what is the value?
- Is the value a primitive(int, bool, etc)?
  - Just put it in the box
- Is it referring (pointing) to an object?
  - Draw arrow to the object in the heap

Heap

What objects are created?

Q2a

Stack

- For each variable what is the value?
- Is the value a primitive(int, bool, etc)?
  - Just put it in the box
- Is it referring (pointing) to an object?
  - Draw arrow to the object in the heap

Heap

What objects are created?

- the object v1 refers to

Q2a

Stack

- For each variable what is the value?
- Is the value a primitive(int, bool, etc)?
  - Just put it in the box
- Is it referring (pointing) to an object?
  - Draw arrow to the object in the heap

Heap

What objects are created?

- the object v1 refers to
- the object v2 refers to

Q2a

Steps:

Q2a

Steps:

- Go through code, create frames/objects/variables when needed

Q2a

Steps:

- Go through code, create frames/objects/variables when needed
  - Method call? create frame on stack

Q2a

Steps:

- Go through code, create frames/objects/variables when needed
  - Method call? create frame on stack
  - Method call finished? Remove stack frame

Q2a

Steps:

- Go through code, create frames/objects/variables when needed
  - Method call? create frame on stack
  - Method call finished? Remove stack frame
  - `new` keyword? create object instance on heap

Q2a

Steps:

- Go through code, create frames/objects/variables when needed
  - Method call? create frame on stack
  - Method call finished? Remove stack frame
  - `new` keyword? create object instance on heap
  - update variables/fields when needed

Q2a

Steps:

- Go through code, create frames/objects/variables when needed
  - Method call? create frame on stack
  - Method call finished? Remove stack frame
  - `new` keyword? create object instance on heap
  - update variables/fields when needed
- Final result at line A is the stack and heap diagram

Q2b

Supposed the representation of `x` and `y` have been changed to a `double` array

```
class Vector2D {
  private double x;
  private double y;
  :
}
```

How would things change?

Q2b

Supposed the representation of `x` and `y` have been changed to a `double` array

```
class Vector2D {
  private double[] coord2D;
  :
}
```

How would things change?

Q2b

Just go through wherever `x` and `y` is used and update accordingly

Q2b

Just go through wherever `x` and `y` is used and update accordingly

- Note that for add there's 2 ways of doing

Q2b

Just go through wherever `x` and `y` is used and update accordingly

- Note that for add there's 2 ways of doing
  - Create a new array

Q2b

Just go through wherever `x` and `y` is used and update accordingly

- Note that for add there's 2 ways of doing
  - Create a new array
  - Update the existing array

```
class Vector2D {
  private double x;
  private double y;

  public Vector2D(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public void add(Vector2D v) {
    this.x = this.x + v.x;
    this.y = this.y + v.y;
    // line A
  }
}
```

```
class Vector2D {
  private double[] coord2D;

  public Vector2D(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public void add(Vector2D v) {
    this.x = this.x + v.x;
    this.y = this.y + v.y;
    // line A
  }
}
```

```
class Vector2D {
  private double[] coord2D;

  public Vector2D(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public void add(Vector2D v) {
    this.x = this.x + v.x;
    this.y = this.y + v.y;
    // line A
  }
}
```

```
class Vector2D {
  private double[] coord2D;

  public Vector2D(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public void add(Vector2D v) {
    this.x = this.x + v.x;
    this.y = this.y + v.y;
    // line A
  }
}
```

```
1  class Vector2D {
2    private double[] coord2D;
3
4    public Vector2D(double x, double y) {
5      this.coord2D = new double[] {x, y};
6    }
7
8    public void add(Vector2D v) {
9      this.x = this.x + v.x;
10     this.y = this.y + v.y;
11     // line A
12   }
13 }
```

```
class Vector2D {
  private double[] coord2D;

  public Vector2D(double x, double y) {
    this.coord2D = new double[] {x, y};
  }

  public void add(Vector2D v) {
    this.x = this.x + v.x;
    this.y = this.y + v.y;
    // line A
  }
}
```

```
1  class Vector2D {
2    private double[] coord2D;
3
4    public Vector2D(double x, double y) {
5      this.coord2D = new double[] {x, y};
6    }
7
8    public void add(Vector2D v) {
9      this.x = this.x + v.x;
10     this.y = this.y + v.y;
11     // line A
12   }
13 }
```

```
1  class Vector2D {
2    private double[] coord2D;
3
4    public Vector2D(double x, double y) {
5      this.coord2D = new double[] {x, y};
6    }
7
8    public void add(Vector2D v) {
9      this.coord2D[0] += v.coord2D[0];
10     this.coord2D[1] += v.coord2D[1];
11     // line A
12   }
13 }
```

```
1  class Vector2D {
2    private double[] coord2D;
3
4    public Vector2D(double x, double y) {
5      this.coord2D = new double[] {x, y};
6    }
7
8    public void add(Vector2D v) {
9      this.x = this.x + v.x;
10     this.y = this.y + v.y;
11     // line A
12   }
13 }
```

```
class Vector2D {
  private double[] coord2D;

  public Vector2D(double x, double y) {
    this.coord2D = new double[] {x, y};
  }

  public void add(Vector2D v) {
    this.coord2D = new double[] {
      this.coord2D[0] + v.coord2D[0],
      this.coord2D[1] + v.coord2D[1]
    }
    // line A
  }
}
```

```
class Vector2D {
  private double[] coord2D;

  public Vector2D(double x, double y) {
    this.coord2D = new double[] {x, y};
  }

  public void add(Vector2D v) {
    this.coord2D = new double[] {
      this.coord2D[0] + v.coord2D[0],
      this.coord2D[1] + v.coord2D[1]
    }
    // line A
  }
}
```

Q2b

Would the program fragment (in `main`) still be valid?

```
Vector2D v1 = new Vector2D(1, 1);
Vector2D v2 = new Vector2D(2, 2);
v1.add(v2);
```

Q2b

Would the program fragment (in `main`) still be valid?

```
Vector2D v1 = new Vector2D(1, 1);
Vector2D v2 = new Vector2D(2, 2);
v1.add(v2);
```

- Yes, all the changes are "internal"

Q2b

Would the program fragment (in `main`) still be valid?

- Yes, all the changes are "internal"
- client on the "outside" doesn't see the changes

```
Vector2D v1 = new Vector2D(1, 1);
Vector2D v2 = new Vector2D(2, 2);
v1.add(v2);
```

Q2b

Would the program fragment (in `main`) still be valid?

```
Vector2D v1 = new Vector2D(1, 1);
Vector2D v2 = new Vector2D(2, 2);
v1.add(v2);
```

- Yes, all the changes are "internal"
- client on the "outside" doesn't see the changes
- Hidden behind abstraction barrier

Q3a

Study the following code

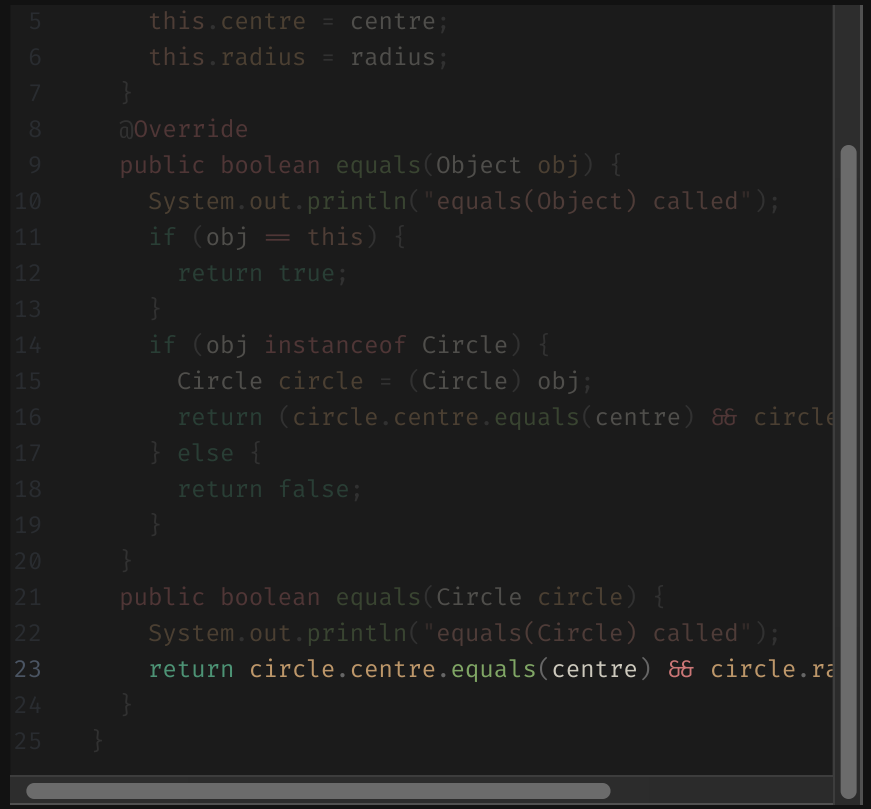
```java
public class Point {
  private double x;
  private double y;
  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

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

Q3a

With the following code fragment, what is the return value of `c1.equals(c2)`

```
Circle c1 = new Circle(new Point(0, 0), 10);
Circle c2 = new Circle(new Point(0, 0), 10);
Object o1 = c1;
Object o2 = c2;
```



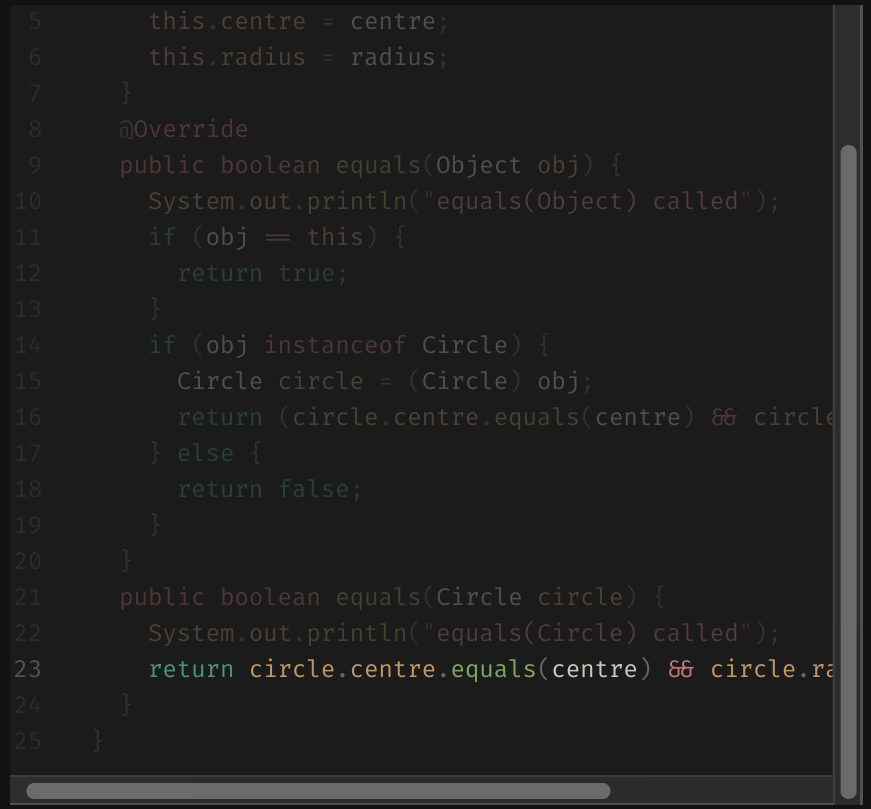
```
5         this.centre = centre;
6         this.radius = radius;
7       }
8       @Override
9       public boolean equals(Object obj) {
10        System.out.println("equals(Object) called");
11        if (obj == this) {
12          return true;
13        }
14        if (obj instanceof Circle) {
15          Circle circle = (Circle) obj;
16          return (circle.centre.equals(centre) && circle
17        } else {
18          return false;
19        }
20      }
21      public boolean equals(Circle circle) {
22        System.out.println("equals(Circle) called");
23        return circle.centre.equals(centre) && circle.ra
24      }
25    }
```

Q3a

With the following code fragment, what is the return value of `c1.equals(c2)`

```
Circle c1 = new Circle(new Point(0, 0), 10);
Circle c2 = new Circle(new Point(0, 0), 10);
Object o1 = c1;
Object o2 = c2;
```

- Returns false

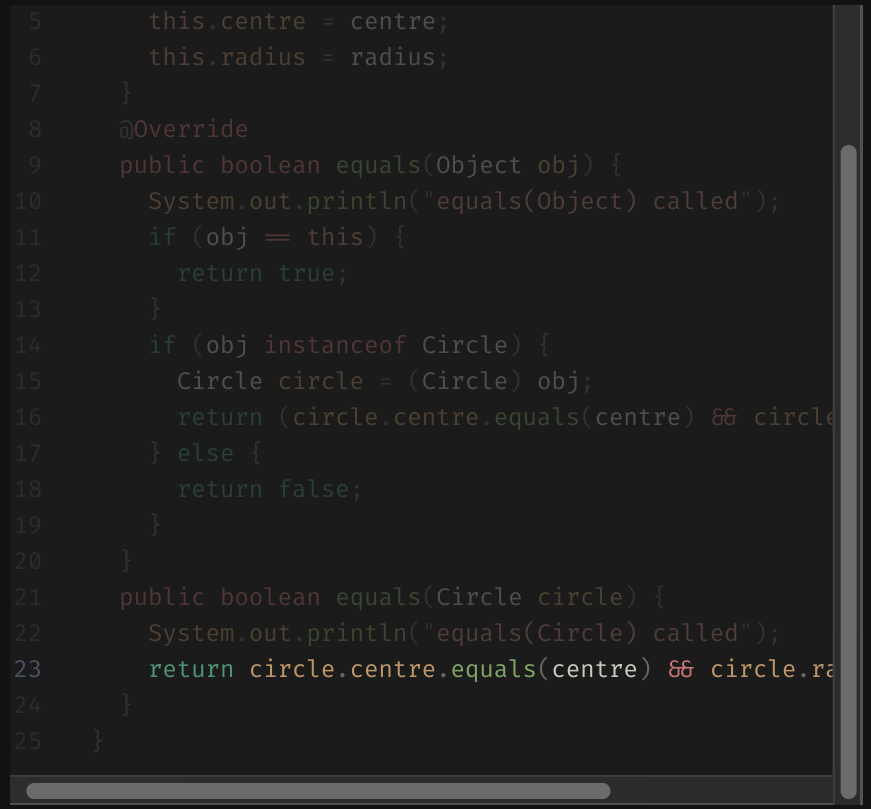


Q3a

With the following code fragment, what is the return value of `c1.equals(c2)`

```
Circle c1 = new Circle(new Point(0, 0), 10);
Circle c2 = new Circle(new Point(0, 0), 10);
Object o1 = c1;
Object o2 = c2;
```

- Returns false
- "same" center, but it's actually 2 diff `Point` instances (brian draw diagram)

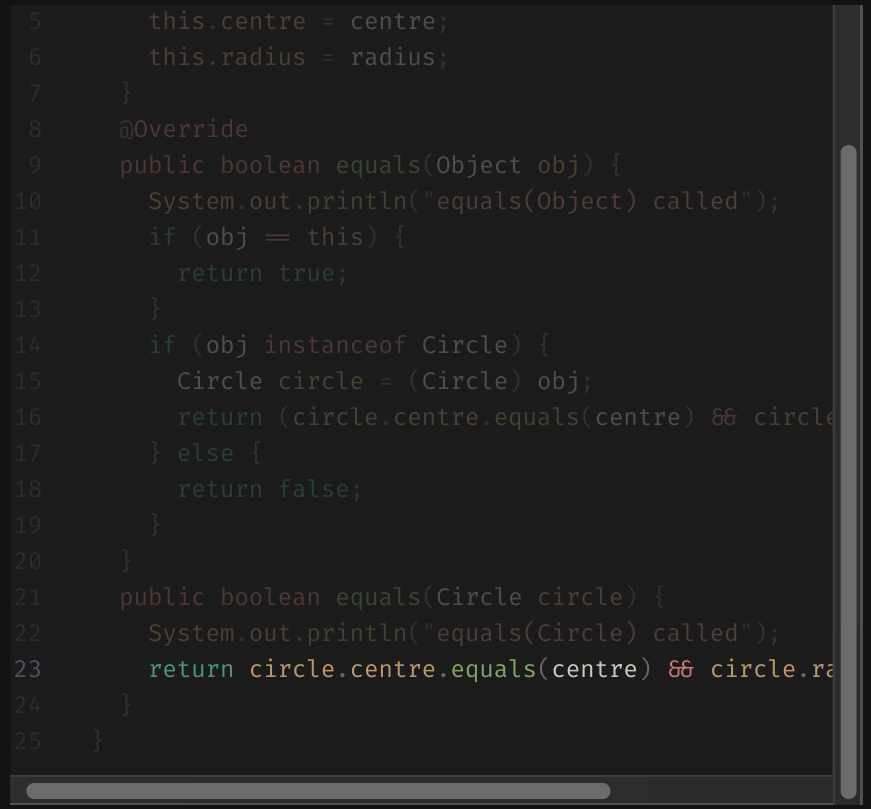


Q3a

With the following code fragment, what is the return value of `c1.equals(c2)`

```
Circle c1 = new Circle(new Point(0, 0), 10);
Circle c2 = new Circle(new Point(0, 0), 10);
Object o1 = c1;
Object o2 = c2;
```

- Returns false
- "same" center, but it's actually 2 diff `Point` instances (brian draw diagram)
- `circle.centre.equals(centre)` is false

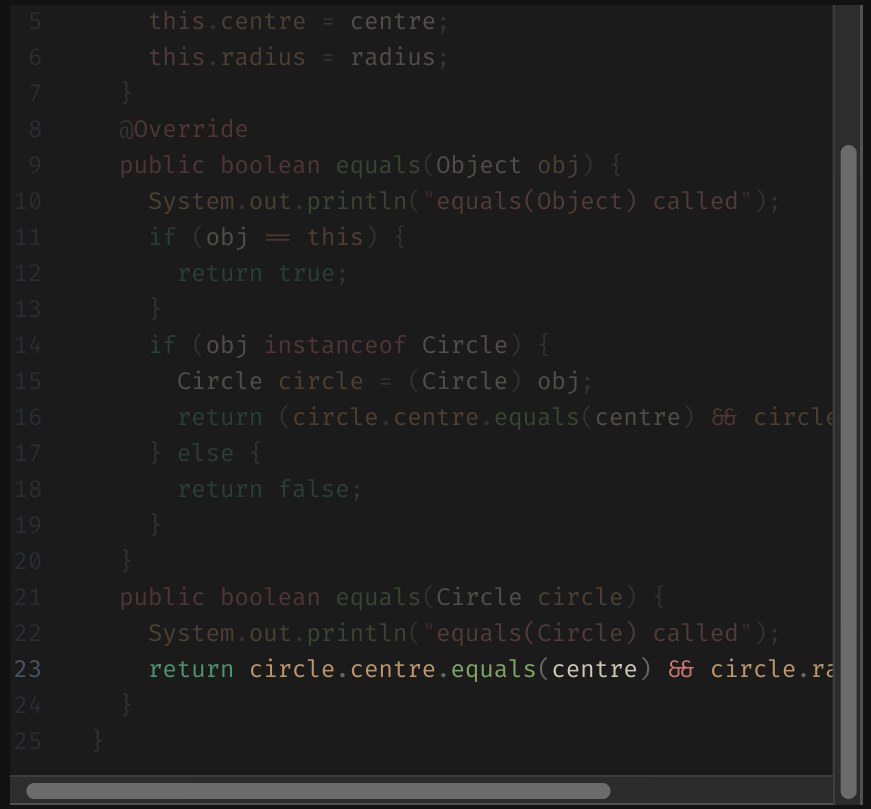


Q3a

With the following code fragment, what is the return value of `c1.equals(c2)`

```
Circle c1 = new Circle(new Point(0, 0), 10);
Circle c2 = new Circle(new Point(0, 0), 10);
Object o1 = c1;
Object o2 = c2;
```

- Returns false
- "same" center, but it's actually 2 diff `Point` instances (brian draw diagram)
- `circle.centre.equals(centre)` is false
- Default `Object::equals` is only true iff same EXACT instance of object
  - Reminder: if starts with capital, classname. start with small letter is instance



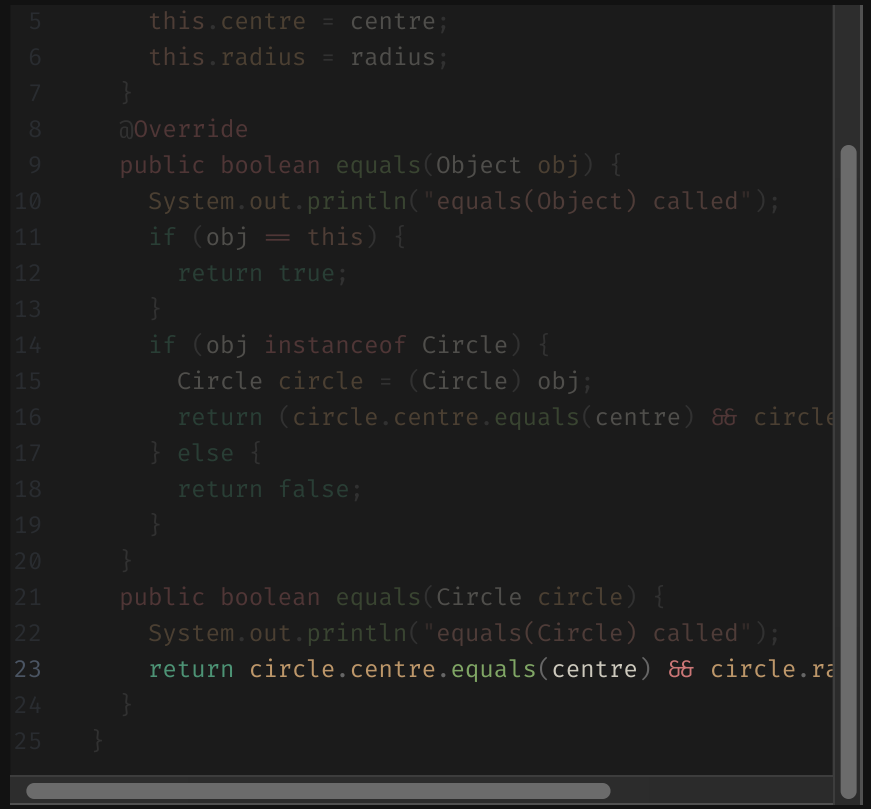
```
5       this.centre = centre;
6       this.radius = radius;
7     }
8     @Override
9     public boolean equals(Object obj) {
10      System.out.println("equals(Object) called");
11      if (obj == this) {
12        return true;
13      }
14      if (obj instanceof Circle) {
15        Circle circle = (Circle) obj;
16        return (circle.centre.equals(centre) && circle
17      } else {
18        return false;
19      }
20    }
21    public boolean equals(Circle circle) {
22      System.out.println("equals(Circle) called");
23      return circle.centre.equals(centre) && circle.ra
24    }
25  }
```

Q3a

With the following code fragment, what is the return value of `c1.equals(c2)`

```
Circle c1 = new Circle(new Point(0, 0), 10);
Circle c2 = new Circle(new Point(0, 0), 10);
Object o1 = c1;
Object o2 = c2;
```

- Returns false
- "same" center, but it's actually 2 diff `Point` instances (brian draw diagram)
- `circle.centre.equals(centre)` is false
- Default `Object::equals` is only true iff same EXACT instance of object
  - Reminder: if starts with capital, classname. start with small letter is instance
- How can we make this return true?



Thank you

See y’all next week :)