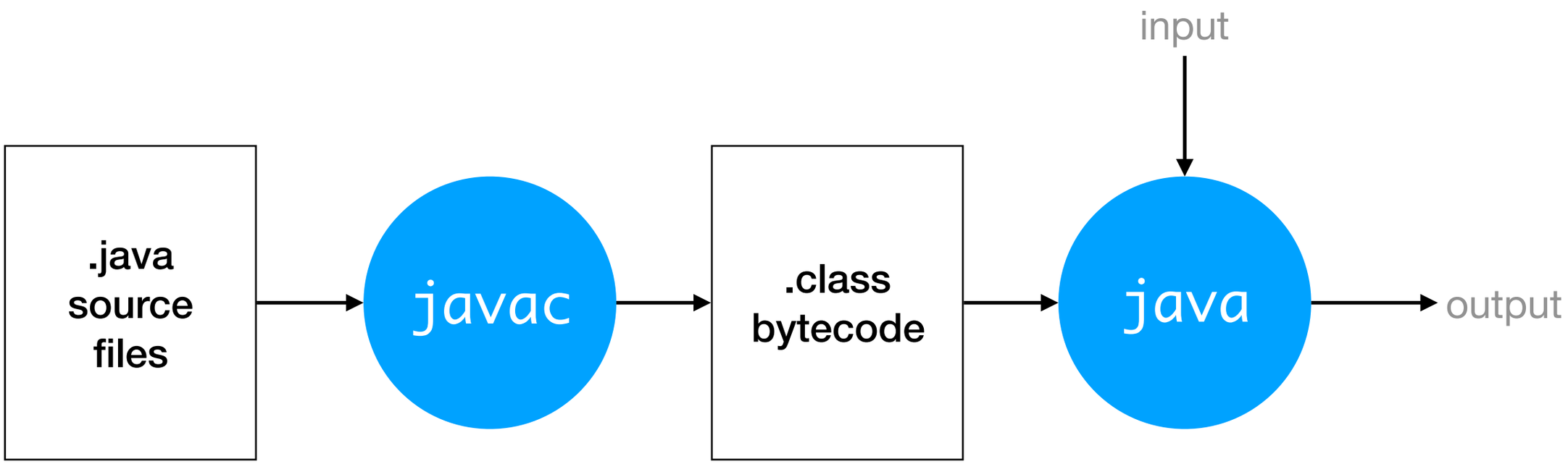
Lecture 4

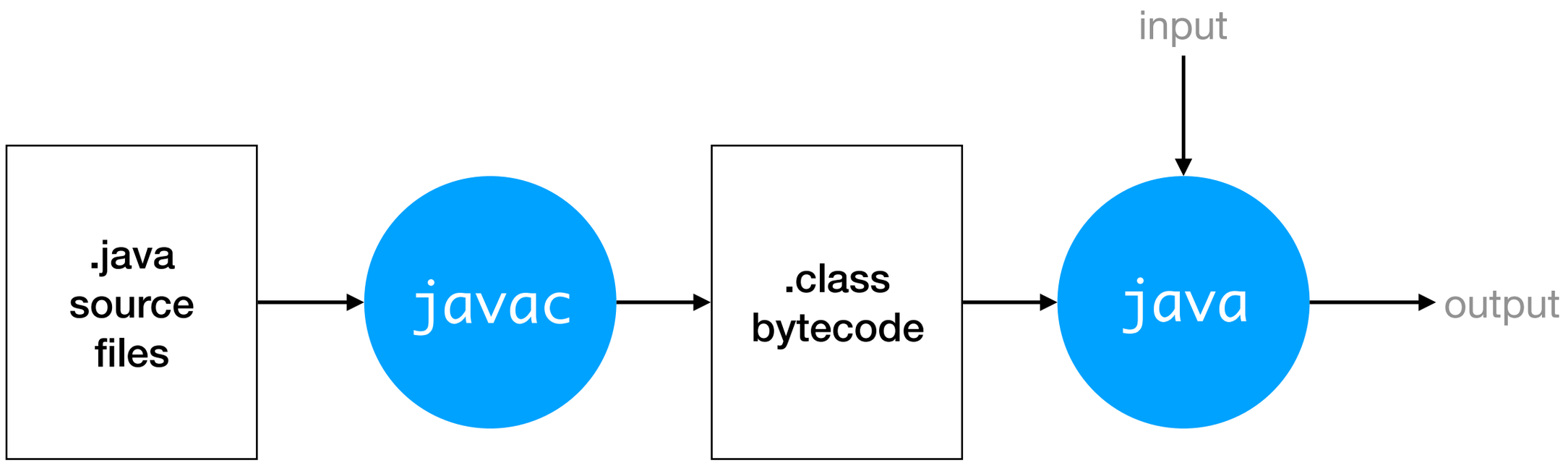
Types, Memory, Exceptions



compile

execute with
Java
Virtual Machine

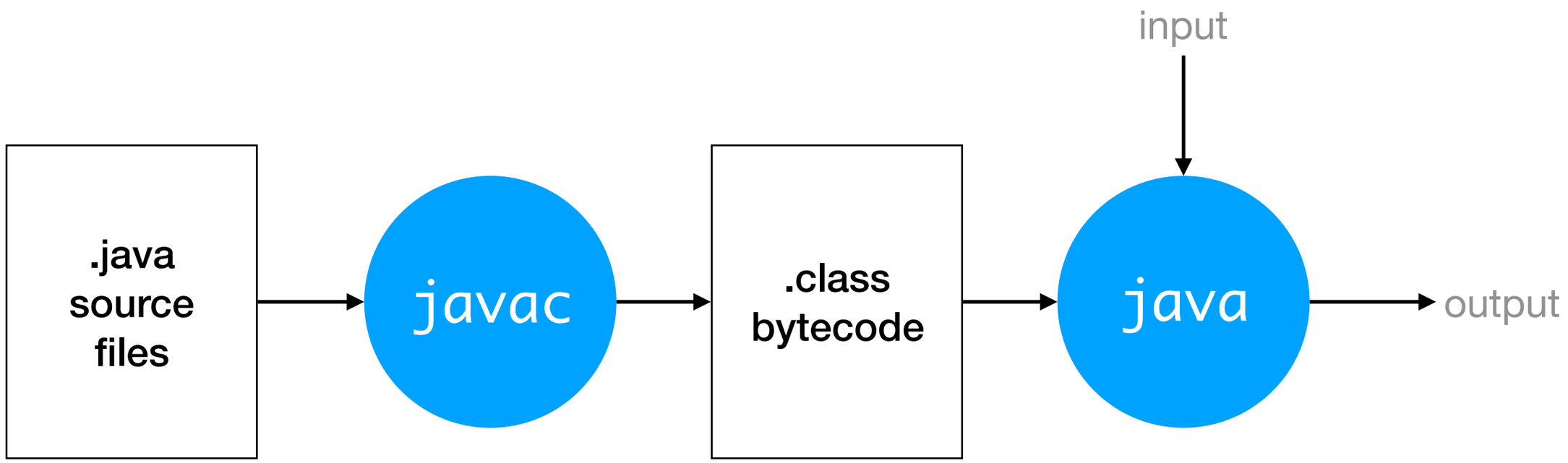
```
for (Printable o: objs) {
   o.print();
}
```



“o is a printable”
“which print() should I call?
don’t know yet..”

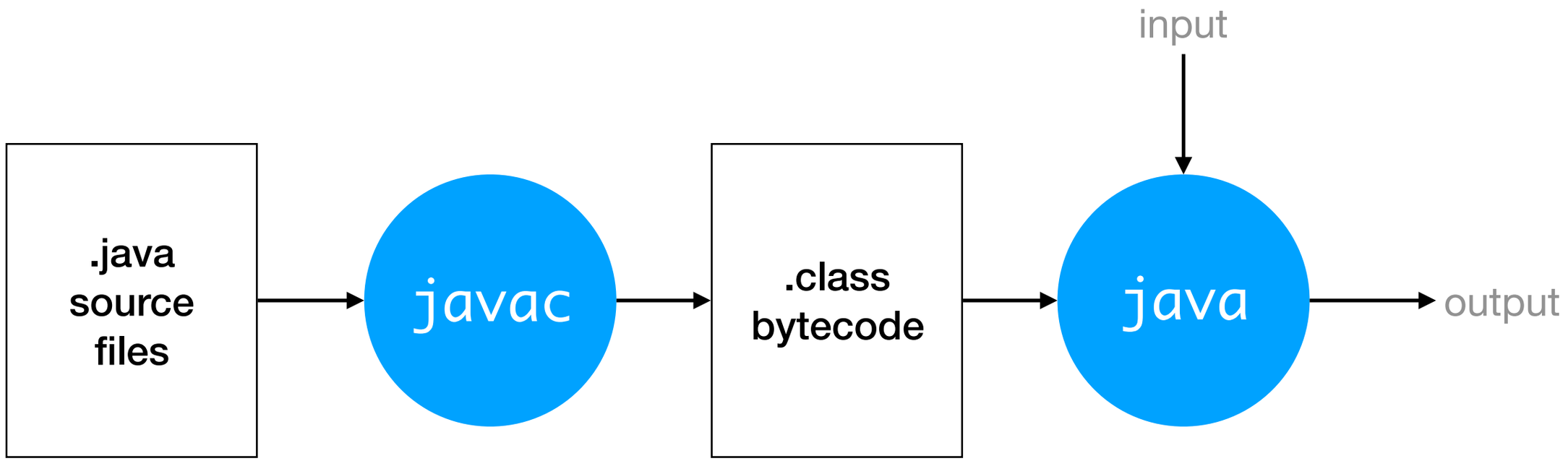
“ok, o is a Point object.
Call this implementation
of print()”

```
Circle c = new Circle(..);
Printable p = c;
p.getArea();
```



“p is a printable”
“error!”

```
Circle c = new Circle(..);
Printable p = c;
((Circle) p).getArea();
```



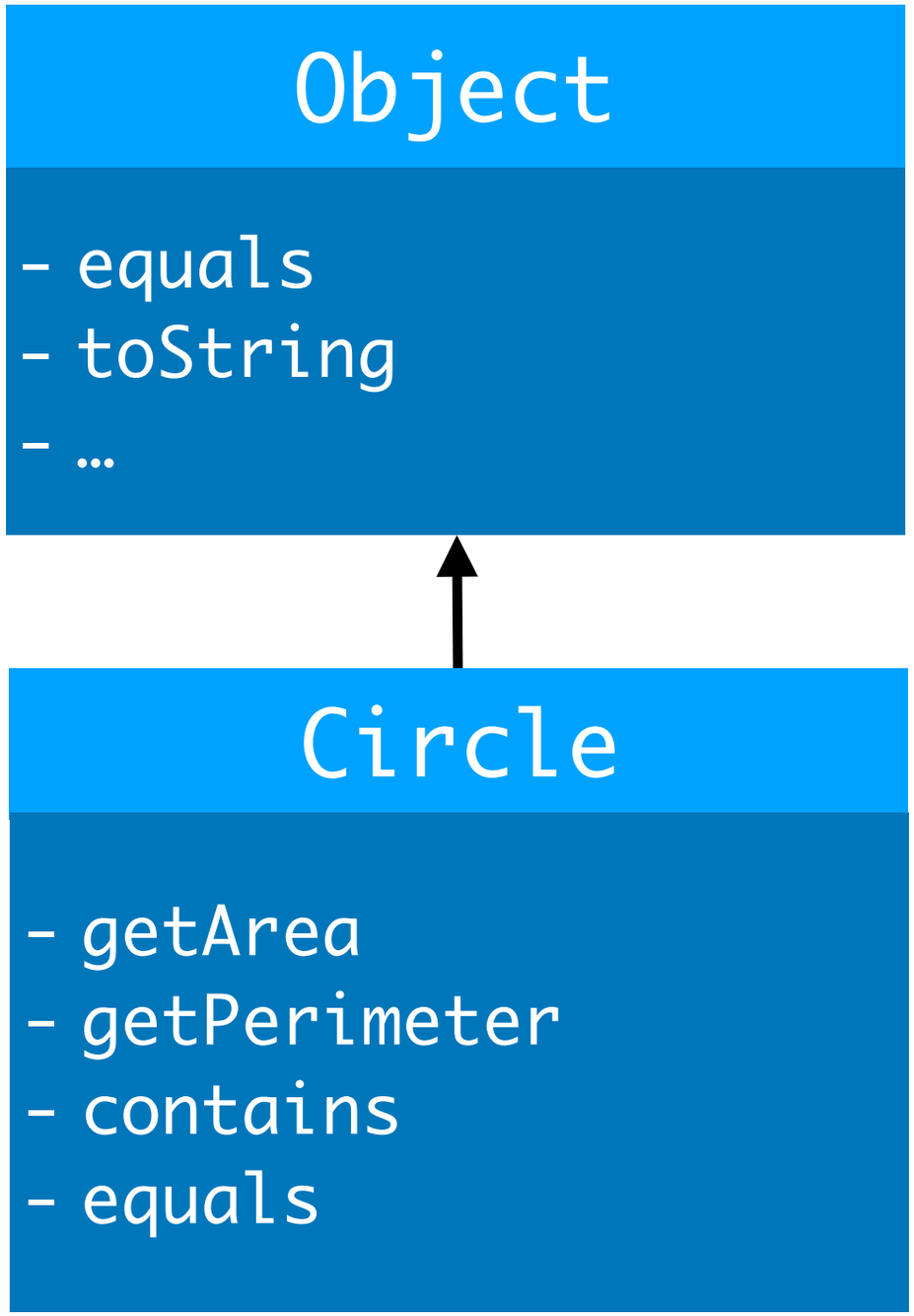
"ok, I hope you know what you are doing.."

```java
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj instanceof Circle) {
        Circle circle = (Circle) obj;
        return (circle.center.equals(center)
                && circle.radius == radius);
    } else
        return false;
}
```

```java
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    Circle circle = obj; // compile error
    return (circle.center.equals(center)
            && circle.radius == radius);
}
```

```java
@Override
public boolean equals(Object obj) {
   if (this == obj) {
      return true;
   }
   Circle circle = (Circle) obj;
   // compile ok
   // but may cause run time error
   return (circle.center.equals(center)
         && circle.radius == radius);
}
```

```java
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj instanceof Circle) {
        Circle circle = (Circle) obj;
        return (circle.center.equals(center)
                && circle.radius == radius);
    } else
        return false;
}
```


Object
- equals
- toString
- …
Circle
- getArea
- getPerimeter
- contains
- equals

```
class Circle implements Shape, Printable
{
        :
        :
}
```

Subtype Relationship

```
Circle <: Shape
Circle <: Printable
Circle <: Object
```

- Widening conversion always OK during compile time and run time
- Narrowing conversion requires casting to compile and may cause run time error

Variance of Types

- Subtype relationship between complex types
- E.g.,
  - is `Circle[]` <: `Object[]`?
  - `equals(Circle)` <: `equals(Object)`?

Covariance

- S <: T means S[ ] <: T[ ]
- Java arrays are covariant
  - is `Circle[]` <: `Object[]`? **Yes**
  - `equals(Circle)` <: `equals(Object)`?
    (no, but that's for another lecture)

- Contravariant: S <: T => T[ ] <: S[ ]
- Bivariant: both co- and contravariant
- Invariant: neither

JVM memory

```
Circle c;
c = new Circle(new Point(1, 1), 8);
```

Heap

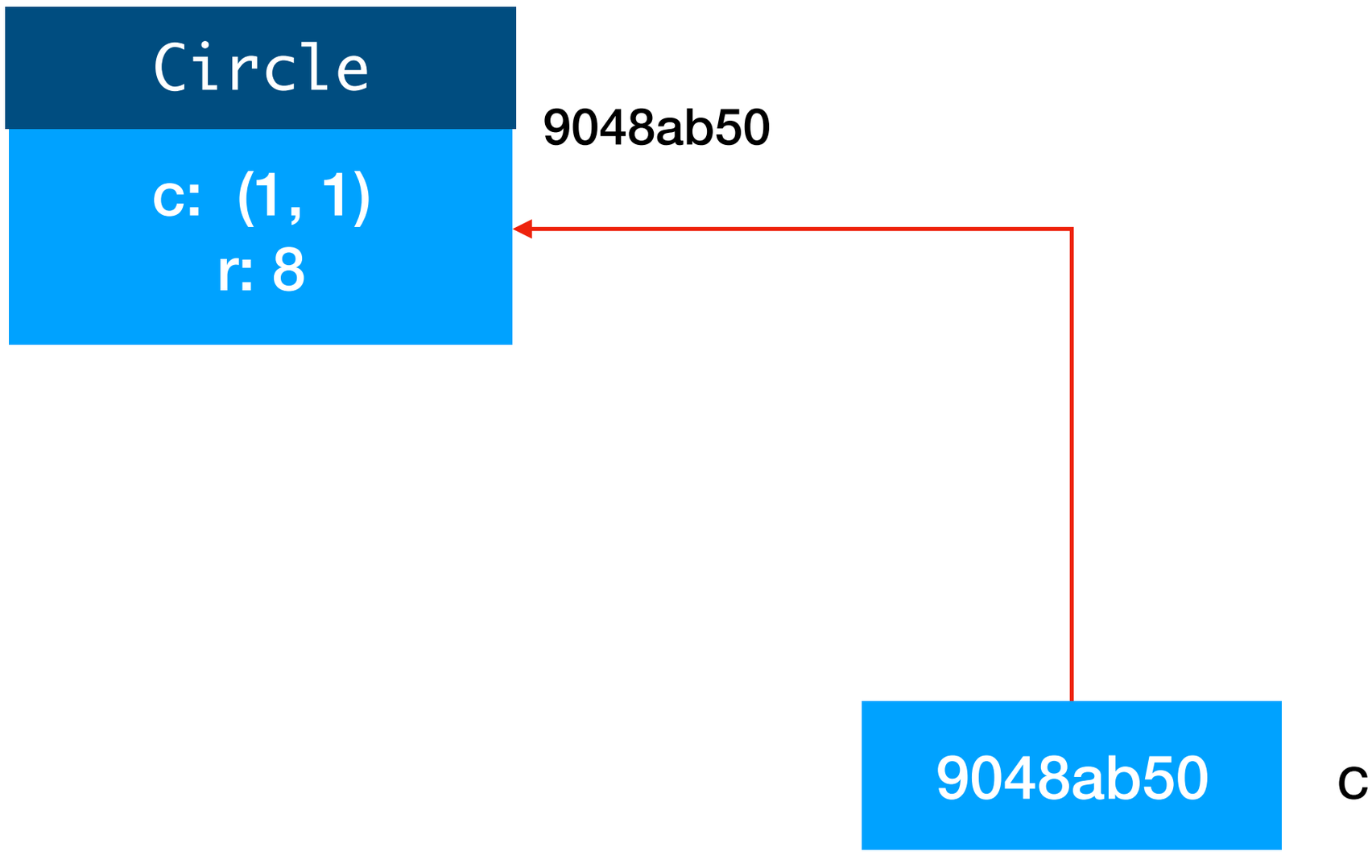
Stack

```
Circle c;
```

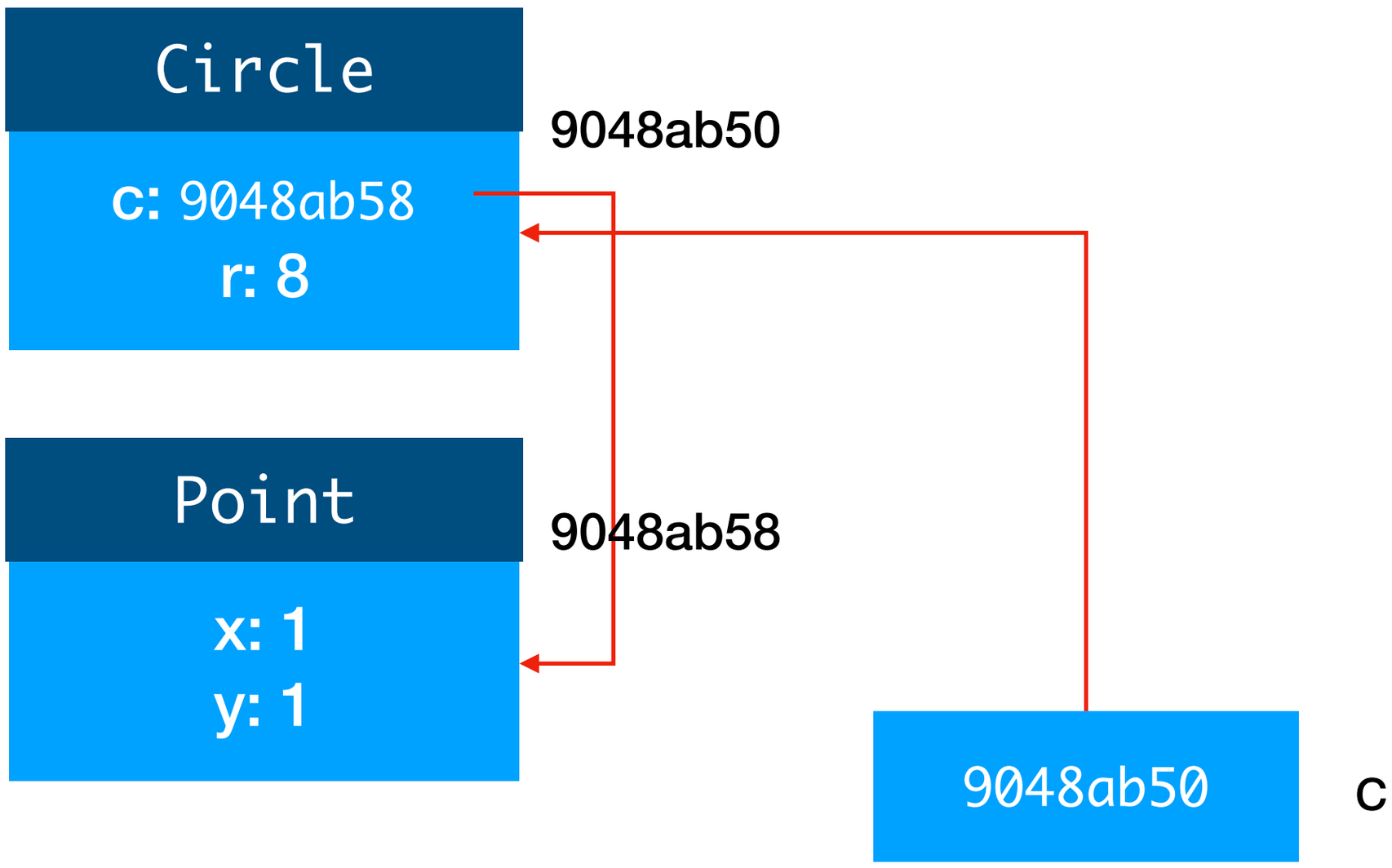
Heap

Stack

```
Circle c;
c = new Circle(new Point(1, 1), 8);
```



```
Circle c;
c = new Circle(new Point(1, 1), 8);
```



```
Circle c;
Point center;
double radius;
```

```
radius = 8;
center = new Point(1, 1);
c = new Circle(center, radius);
```

Heap

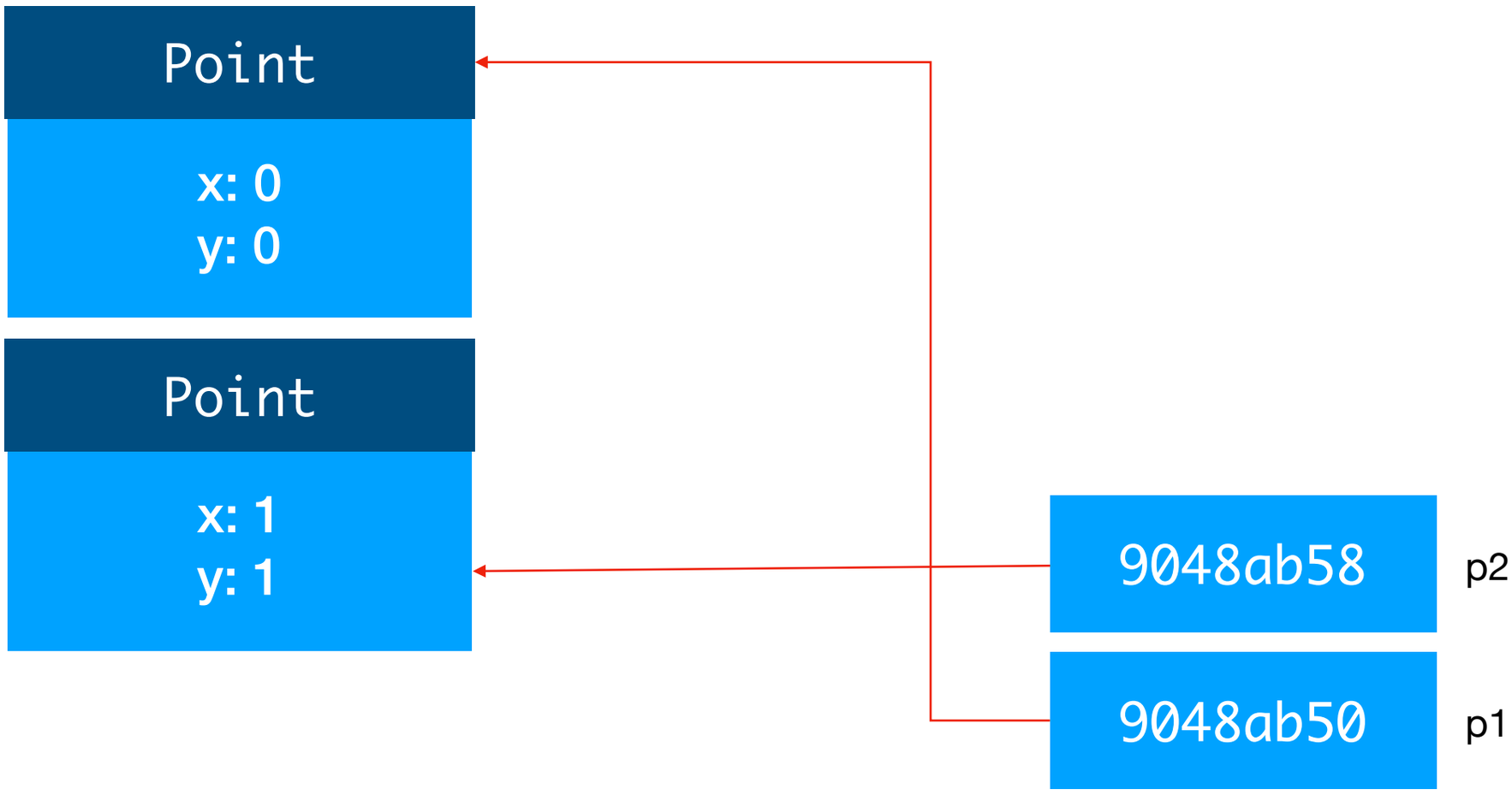
Stack

```
Point p1 = new Point(0,0);
Point p2 = new Point(1,1);
p1.distanceTo(p2);
```

```
public double distanceTo(Point q) {
    return …
}
```

Heap

Stack

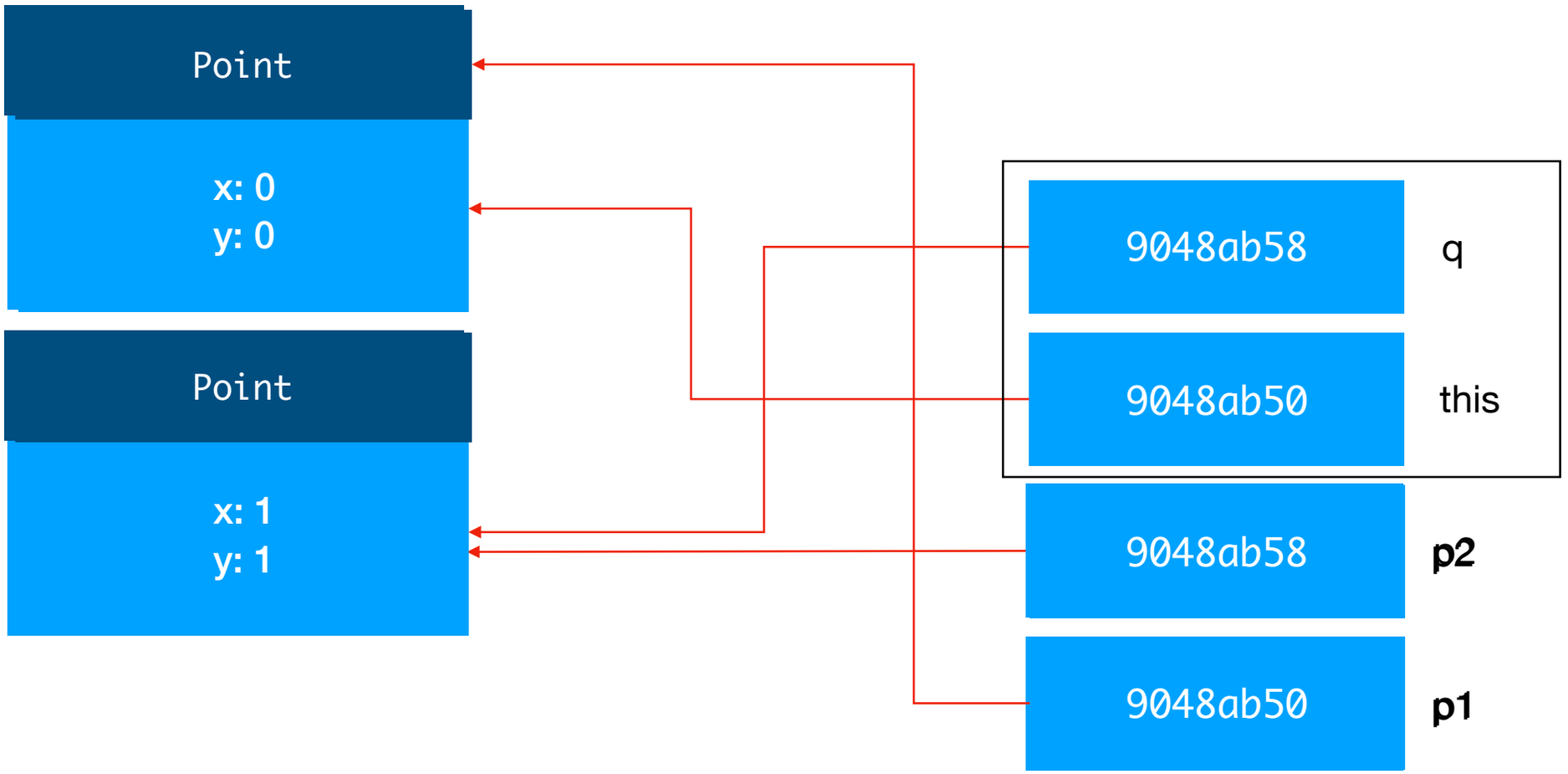


```
Point p1 = new Point(0,0);
Point p2 = new Point(1,1);
p1.distanceTo(p2);
```

```
public double distanceTo(Point q) {
    return …
}
```

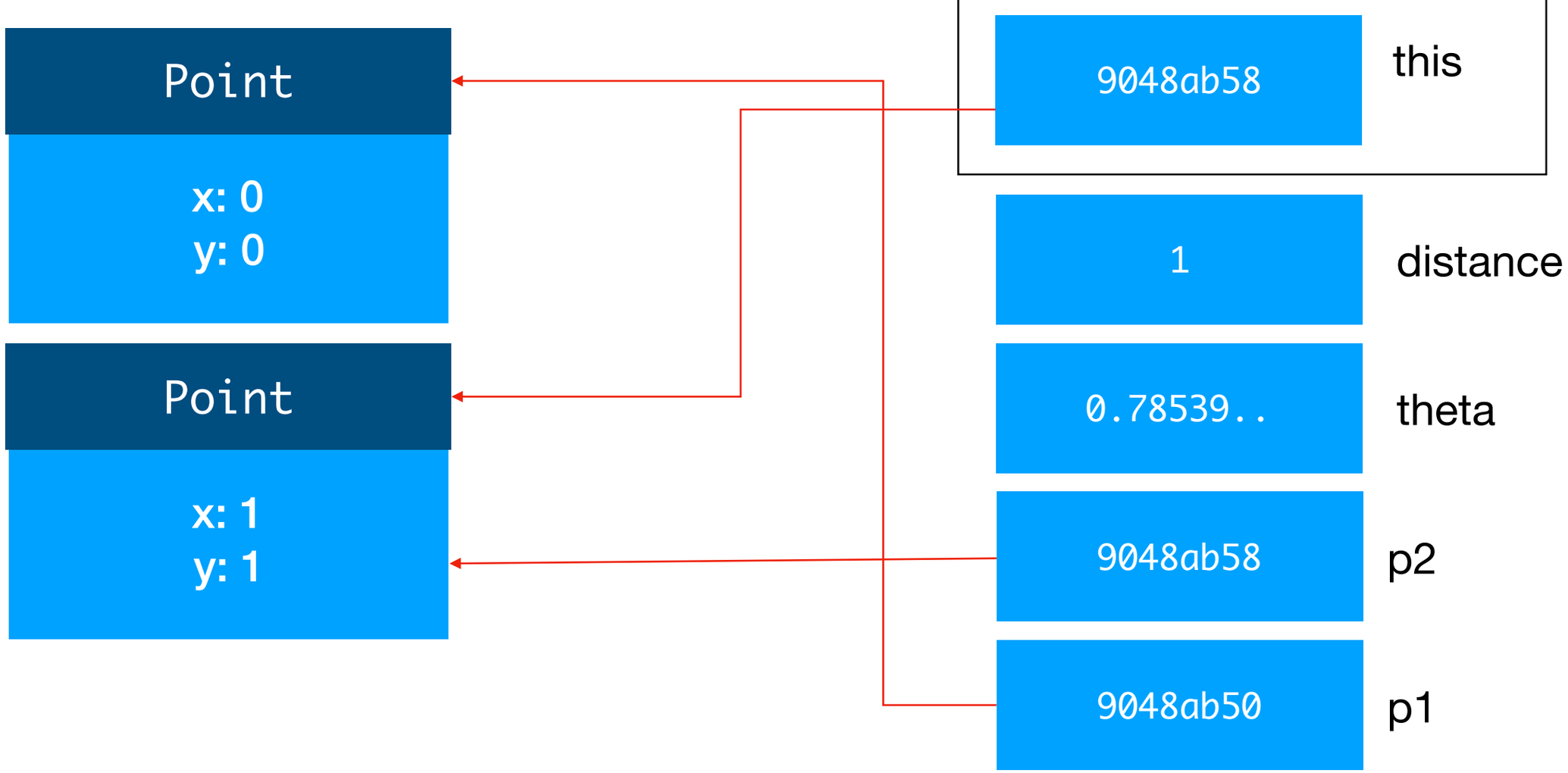
Heap

Stack



```
double theta = Math.PI/4.0;
double distance = 1;
p2.move(theta, distance);
```

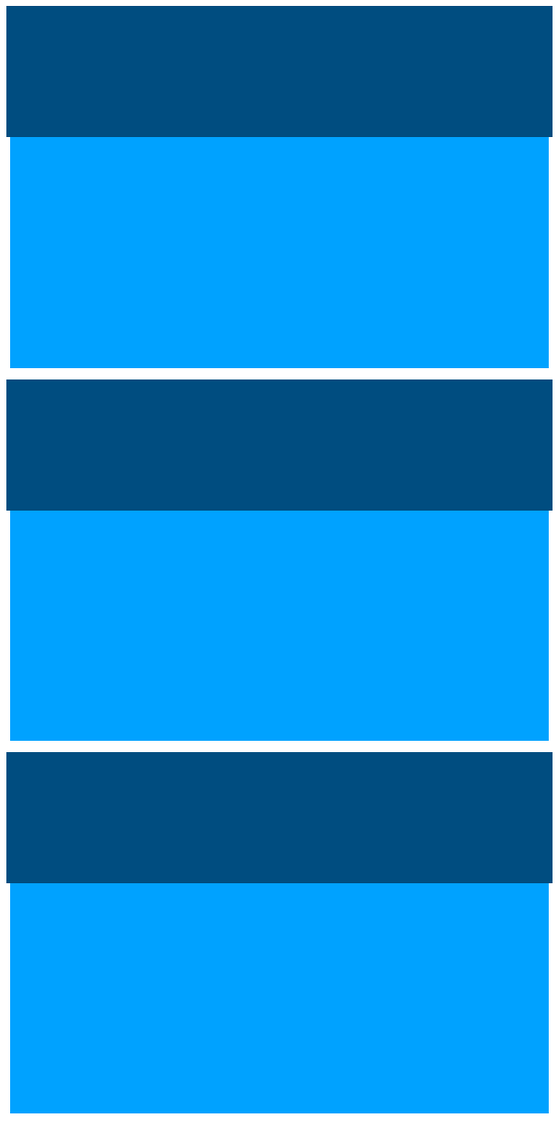
```
public double move(double theta,
  double d) {  …
}
```

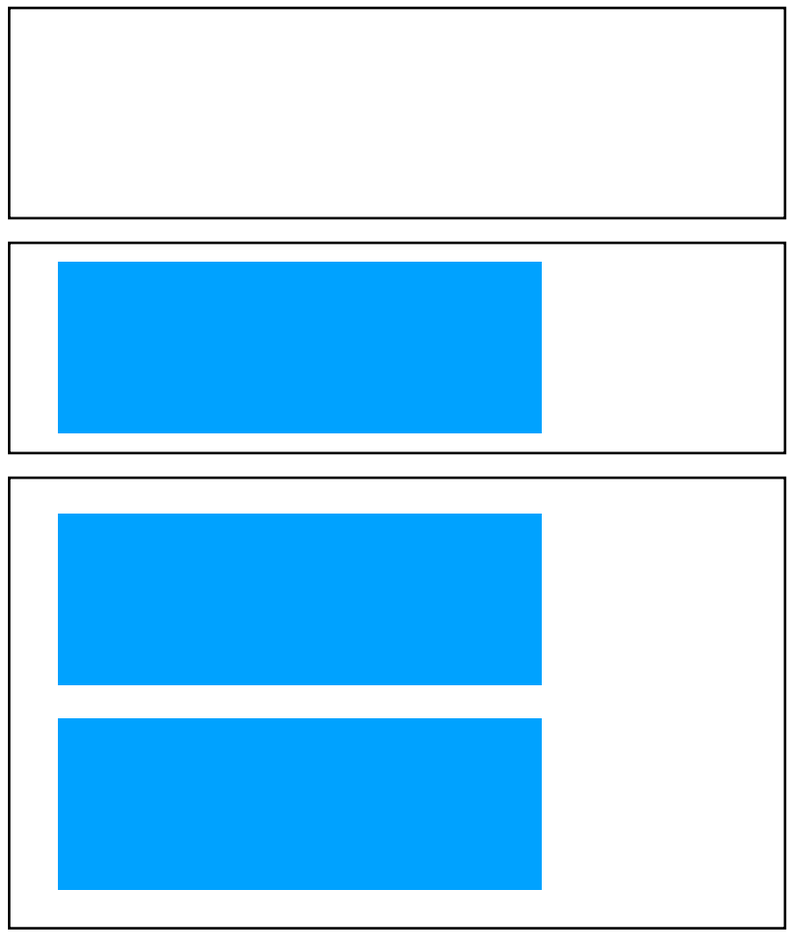


Java uses
call by value for primitive types,
call by reference for objects

Heap

Stack

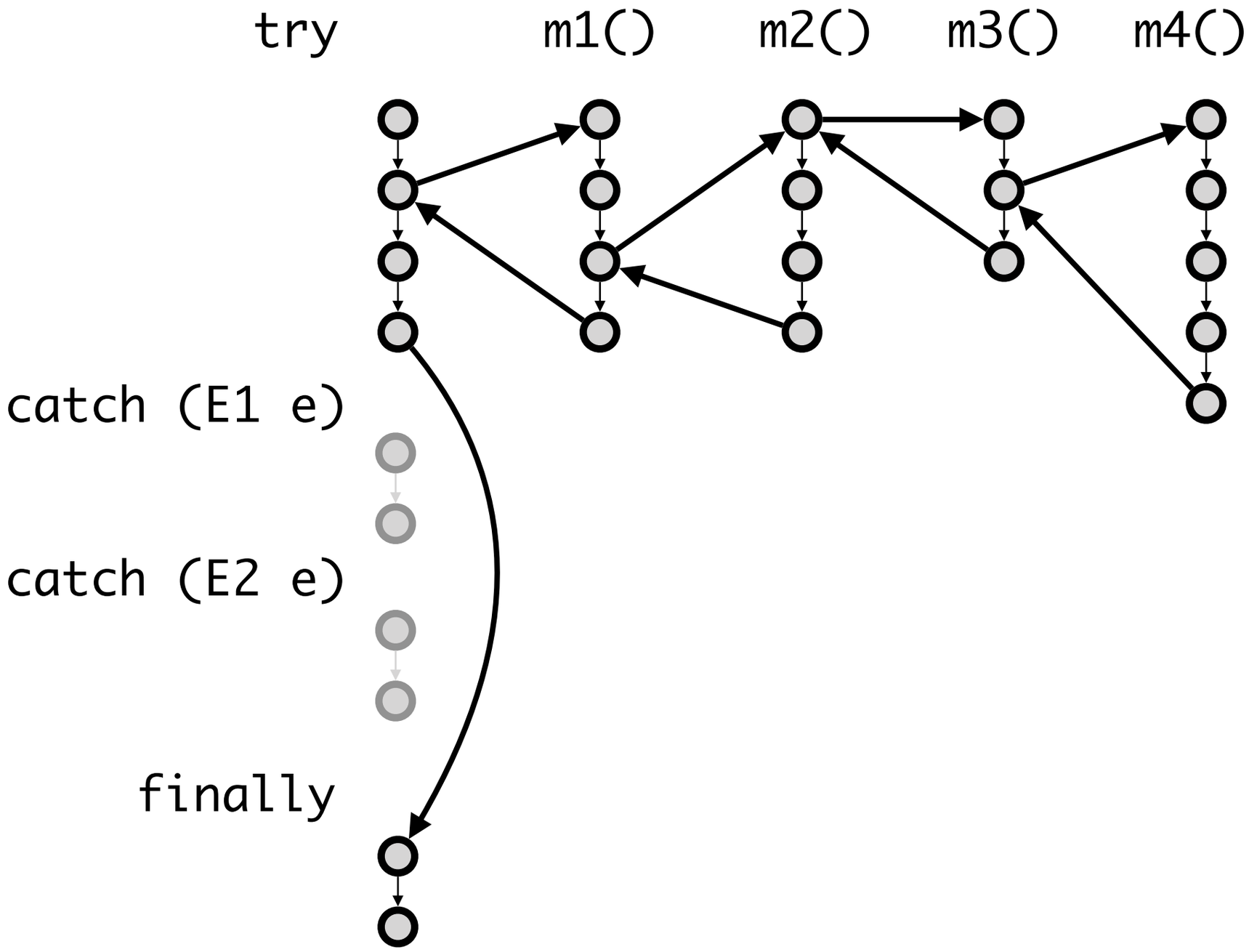


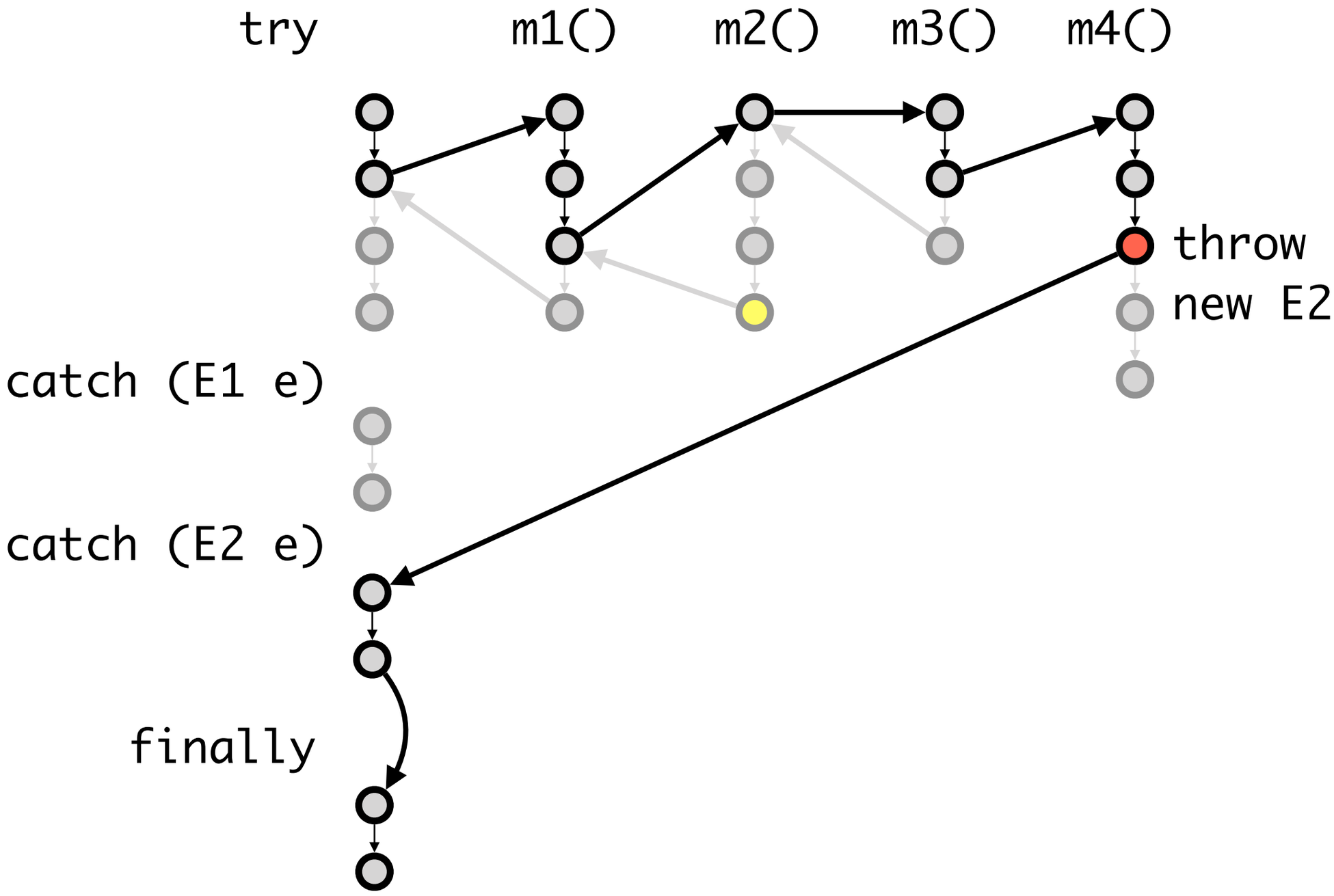


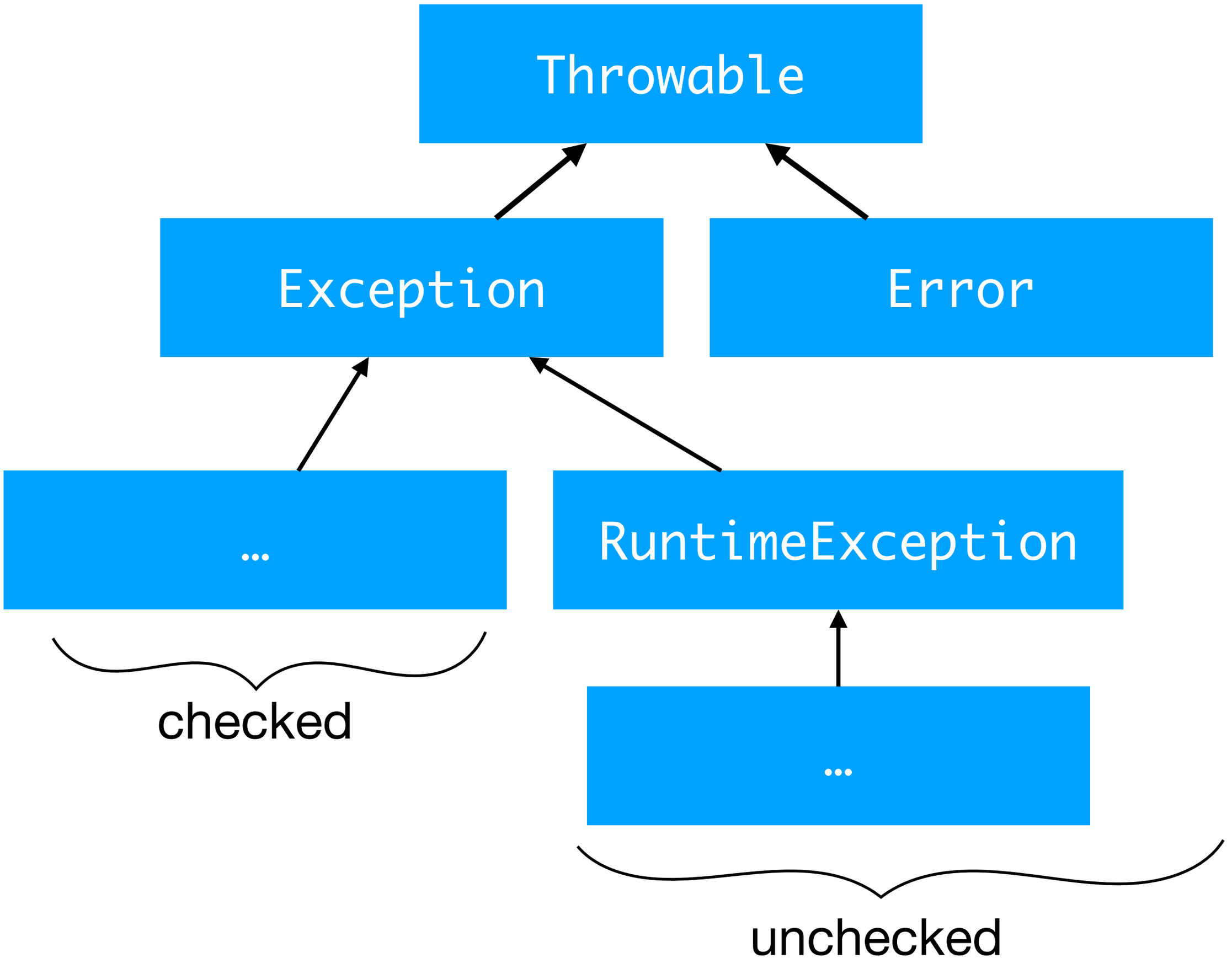
main

Java is a garbage collected language

How to handle errors


try
m1()
m2()
m3()
m4()
catch (E1 e)
catch (E2 e)
finally


try
m1()
m2()
m3()
m4()
throw
new E2
catch (E1 e)
catch (E2 e)
finally


Throwable
Exception
Error
...
RuntimeException
...
checked
unchecked

All checked exception must be caught or thrown

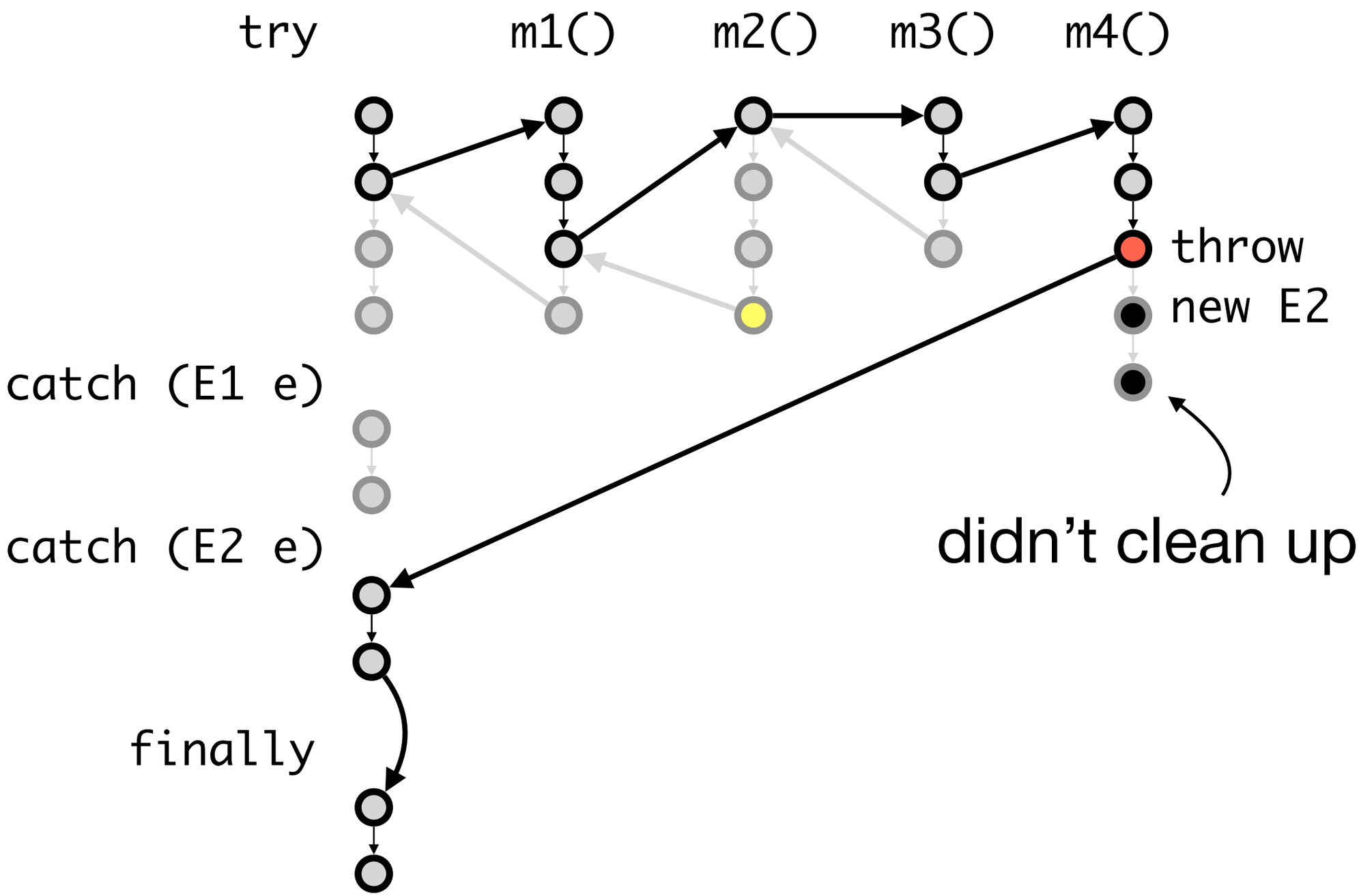
```
public static int readIntFrom(String filename)
  throws FileNotFoundException {
  FileReader reader = new FileReader(filename);
  Scanner scanner = new Scanner(reader);
  int numOfPoints = scanner.nextInt();
  scanner.close();
  return numOfPoints;
}
```

```
public Circle(Point p, Point q, double r) {
   if (p.distanceTo(q) > 2*r) {
      throw new IllegalArgumentException(
       "Input points are too far apart");
   }
   if (p.equals(q)) {
      throw new IllegalArgumentException(
       “Input points coincide");
   }
      :
}
```

Liskov Substitution Principle (LSP)

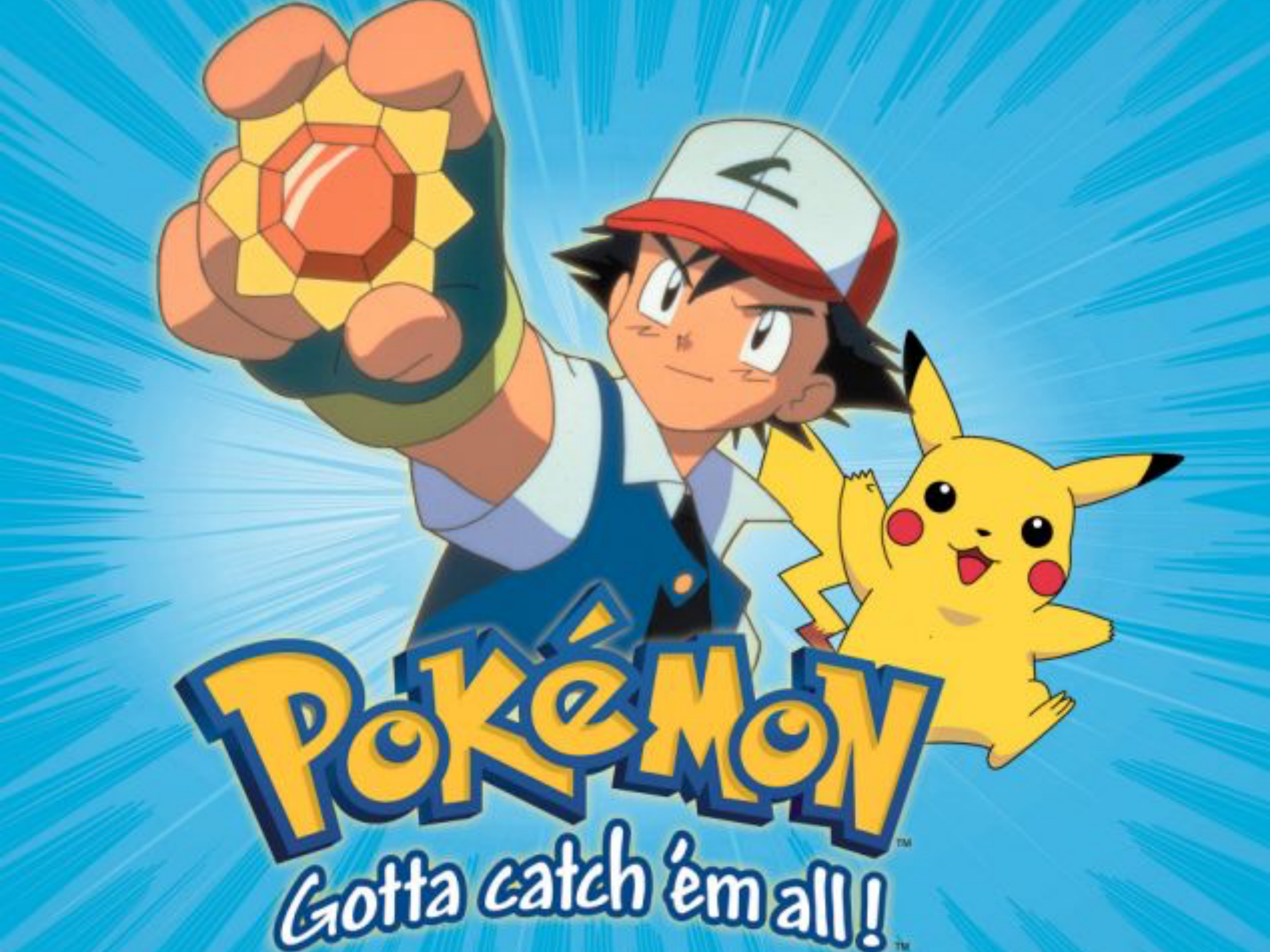
if T is a subclass of S and f() of T overrides f() of S, f() of T cannot throw a more general exception than f() of S.

Exceptions
(done wrong)


try
m1()
m2()
m3()
m4()
throw
new E2
catch (E1 e)
catch (E2 e)
finally
didn’t clean up

```
public static int readIntFrom(String filename)
  throws FileNotFoundException {
  FileReader reader = new FileReader(filename);
  Scanner scanner = new Scanner(reader);
  int numOfPoints = scanner.nextInt();
  scanner.close();
  return numOfPoints;
}
```

```
try {
    // your code
}
catch (Exception e) {
    // handle exception
}
```


POKÉMON
Gotta catch 'em all!

```
try {
   // your code
}
catch (Exception e) {
   System.exit(0);
}
```

```
try {
    // your code
}
catch (Error e) {
    // do nothing
}
```

```
class ClassRoster {
    :
   public Students[] getStudents()
       throws FileNotFoundException {
       :
   }
}
```

```
class ClassRoster {
    :
   public Students[] getStudents()
       throws SQLException {
       :
   }
}
```