Checking Understanding

- Take a look at the classes in the workbook!
- You have `Boat`, `SailBoat`, `MotorBoat`, and `Sinkable`
  - Plus some additional "helper" classes
- **Question 1:** Draw the UML of the code in the attachment
  - You do not need to include the attributes/methods

Checking Understanding (cont.)

(Review 1) What kind of error would be produced?

0

```
// Given:
Boat[] boats = new Boat[3];
Sinkable sink = new SailBoat("TrueSails", 2);
boats[0] = new MotorBoat("TrailBlazer", 3);
boats[1] = new SailBoat("EasyGoing", 1);

// Consider the following code:
boats.move();
```

- Compile-time error
- Runtime error
- No error

Checking Understanding (cont.)

(Review 2) What kind of error would be produced?

0

```
// Given:
Boat[] boats = new Boat[3];
Sinkable sink = new SailBoat("TrueSails", 2);
boats[0] = new MotorBoat("TrailBlazer", 3);
boats[1] = new SailBoat("EasyGoing", 1);

// Consider the following code:
boats[0].move();
```

- Compile-time error
- Runtime error
- No error

Checking Understanding (cont.)

(Review 3) What kind of error would be produced?

0

```
// Given:
Boat[] boats = new Boat[3];
Sinkable sink = new SailBoat("TrueSails", 2);
boats[0] = new MotorBoat("TrailBlazer", 3);
boats[1] = new SailBoat("EasyGoing", 1);

// Consider the following code:
((SailBoat) boats[0]).sail();
```

- Compile-time error
- Runtime error
- No error

Checking Understanding (cont.)

(Review 4) What kind of error would be produced?

0

```
// Given:
Boat[] boats = new Boat[3];
Sinkable sink = new SailBoat("TrueSails", 2);
boats[0] = new MotorBoat("TrailBlazer", 3);
boats[1] = new SailBoat("EasyGoing", 1);

// Consider the following code:
boats[2] = new Boat("JohnB");
```

- Compile-time error
- Runtime error
- No error

Checking Understanding (cont.)

(Review 5) What would this code output?

0

```
// Given:
Boat[] boats = new Boat[3];
Sinkable sink = new SailBoat("TrueSails", 2);
boats[0] = new MotorBoat("TrailBlazer", 3);
boats[1] = new SailBoat("EasyGoing", 1);

// Consider the following code:
Boat b = boats[1];
System.out.println(b.toString());
```

Motorboat: EasyGoing x:0 y:0 Count: 3

Boat: EasyGoing x: 0 y: 0 Count: 3

Boat: EasyGoing 1

Sailboat: EasyGoing x: 0 y: 0 Count: 3

Sailboat: EasyGoing 1

Challenge

- Given these four methods in a class called `BoatUtils`:

```
public static void showBoat(Boat boat) {
    System.out.println("BOAT: " + boat);
}

public static void showBoat(SailBoat boat) {
    System.out.println("SAILBOAT: " + boat);
}

public static void showBoat(MotorBoat boat) {
    System.out.println("MOTORBOAT: " + boat);
}

public static void showBoat(Sinkable boat) {
    System.out.println("SINK: " + boat);
}
```

Take a look! They have the same name, but different *parameter lists*

That's perfectly fine! This is called "method **overloading**"

Challenge (cont.)

(Challenge 1) Which method would be called by this code? 0

```
SailBoat sb = new SailBoat("DynamicWind", 123);
BoatUtils.showBoat(sb);
```

- showBoat(Boat boat)
- showBoat(SailBoat boat)
- showBoat(MotorBoat boat)
- showBoat(Sinkable boat)

Challenge (cont.)

(Challenge 2) Which method would be called by this code? 0

```
MotorBoat mb = new MotorBoat("ChaseMe", 5);
BoatUtils.showBoat(mb);
```

showBoat(Boat boat)

showBoat(SailBoat boat)

showBoat(MotorBoat boat)

showBoat(Sinkable boat)

Challenge (cont.)

(Challenge 3) Which method would be called by this code?

0

```
MotorBoat mb = new MotorBoat("ChaseMe", 5);
Boat b1 = mb;
BoatUtils.showBoat(b1);
```

showBoat(Boat boat)

showBoat(SailBoat boat)

showBoat(MotorBoat boat)

showBoat(Sinkable boat)

Challenge (cont.)

(Challenge 4) Which method would be called by this code? 0

```
SailBoat sb = new SailBoat("DynamicWind", 123);
Boat b2 = sb;
BoatUtils.showBoat(b2);
```

showBoat(Boat boat)

showBoat(SailBoat boat)

showBoat(MotorBoat boat)

showBoat(Sinkable boat)

Challenge Code Output

```
public static void showBoat(Boat boat) {
    System.out.println("BOAT: " + boat);
}

public static void showBoat(SailBoat boat) {
    System.out.println("SAILBOAT: " + boat);
}

public static void showBoat(MotorBoat boat) {
    System.out.println("MOTORBOAT: " + boat);
}

public static void showBoat(Sinkable boat) {
    System.out.println("SINK: " + boat);
}
```

BTW: This will *implicitly* call boat.toString()

- Discuss your answers with a partner and try to make sense of what's happening here!

```
SailBoat sb = new SailBoat("DynamicWind", 123);
BoatUtils.showBoat(sb);

  SAILBOAT: Sailboat: DynamicWind 123

MotorBoat mb = new MotorBoat("ChaseMe", 5);
BoatUtils.showBoat(mb);

  MOTORBOAT: Boat: ChaseMe x: 0 y: 0 Count: 5

Boat b1 = mb;
BoatUtils.showBoat(b1);

  BOAT: Boat: ChaseMe x: 0 y: 0 Count: 5

Boat b2 = sb;
BoatUtils.showBoat(b2);

  BOAT: Sailboat: DynamicWind 123
```

Static vs. Dynamic Binding

Static binding is a **compile-time** determination:

- Based on **variable declaration**

- Assignment can only be done to variable *higher up* in hierarchy
  - ***Example:***
```
Picture pic  = new SlideShow(…); // compiles
```

- Only methods present in the variable type can be called
  - ***Example:***
```
Picture pic  = new SlideShow(…);
pic.toString(); // compiles
pic.getCurrentPic(); // doesn't compile!
```

- *Variable type* determines method when passed as a parameter
  - ***Example:*** two overloaded methods `show(Picture p)` and `show(SlideShow s)`

```
show(pic); // runs show(Picture p) – pic has type Picture
```

Static vs. Dynamic Binding (cont.)

Dynamic binding is a **runtime** determination:

- Based on object/instance type
  - (The class/type on the *right,* after the keyword *new*)

- Determines what actually gets run when called
  - ***Example:***

```
Picture pic  = new SlideShow(…);
pic.toString(); // runs toString() defined in SlideShow
// only runs .toString() in superclass if not overridden in SlideShow
```

- The type of the instance applies to over**ridden** methods
  - Static binding applies to over**loaded** methods

So What's Happening Here?

- Take a look at the output. There's two parts to this puzzle!

```
public static void showBoat(Boat boat) {
    System.out.println("BOAT: " + boat);
}

public static void showBoat(SailBoat boat) {
    System.out.println("SAILBOAT: " + boat);
}

public static void showBoat(MotorBoat boat) {
    System.out.println("MOTORBOAT: " + boat);
}

public static void showBoat(Sinkable boat) {
    System.out.println("SINK: " + boat);
}
```

1) 2)

```
SailBoat sb = new SailBoat("DynamicWind", 123);
Boat b2 = sb;
BoatUtils.showBoat(b2);
```

Output:

1) `showBoat(Boat boat)` is called, because the type of the `b2` variable is `Boat`

```
BOAT: Sailboat: DynamicWind 123
```

2) But when we call `.toString()`, we "go to" the object referenced by `b2` (which is a `SailBoat`) and ask *it* to give us a `String`

Test Your Knowledge

(Test 1) Which method would be called by this code?

0

```
Sinkable sk = new MotorBoat("ZippyDriver", 200);
BoatUtils.showBoat(sk);
```

showBoat(Boat boat)

showBoat(SailBoat boat)

showBoat(MotorBoat boat)

showBoat(Sinkable boat)

Test Your Knowledge

(Test 2) What would this code output?

0

```
public static void showBoat(Boat boat) {
    System.out.println("BOAT: " + boat);
}

public static void showBoat(SailBoat boat) {
    System.out.println("SAILBOAT: " + boat);
}

public static void main(String[] args) {
    // What would this code output?
    Boat bt = new SailBoat("CravinSpeed", 200);
    BoatUtils.showBoat(bt);
}
```

SAILBOAT: Sailboat: CravinSpeed 200

SAILBOAT: Boat: CravinSpeed x: 0 y: 0 Count: 1

BOAT: Sailboat: CravinSpeed 200

BOAT: Boat: CravinSpeed x: 0 y: 0 Count: 1