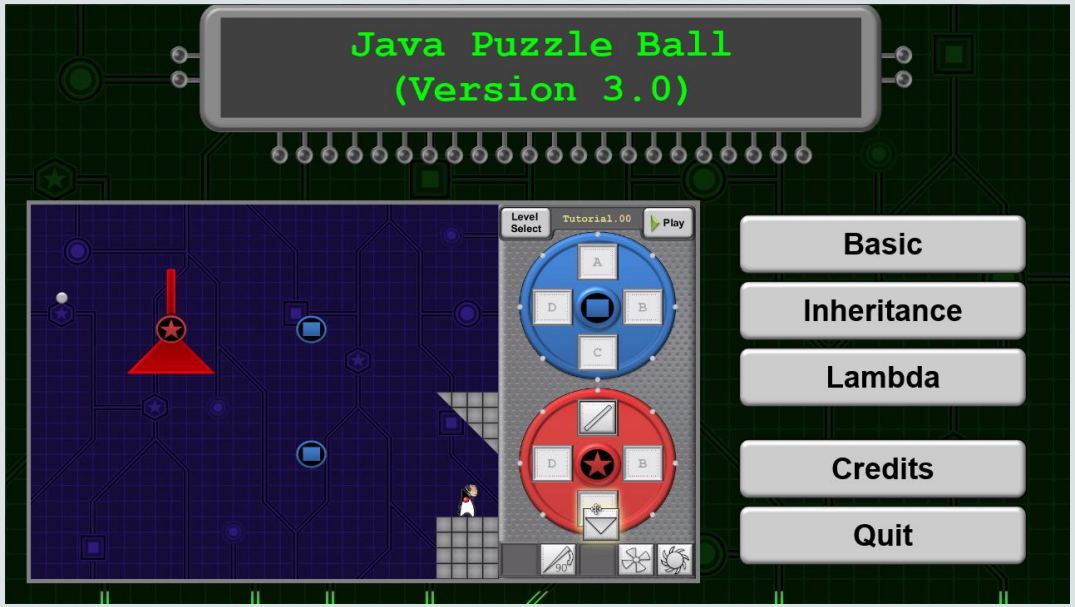

Java™
ORACLE®

Java Puzzle Ball

Nick Ristuccia

Lesson 3-3
Editing Java Code



Remember Methods and Their Implementation

This method is called **`methodA()`**

```
public class BlueBumper {

…
    public void methodA(){
        triangleWall();
    }
    public void methodB(){
        simpleWall();
    }
    public void methodC(){

    }
    public void methodD(){

    }
}
```

Its implementation exists between two curly braces **`{ }`**

The complex implementation is represented by **`triangleWall()`**

Java Puzzle Ball Code is Complex

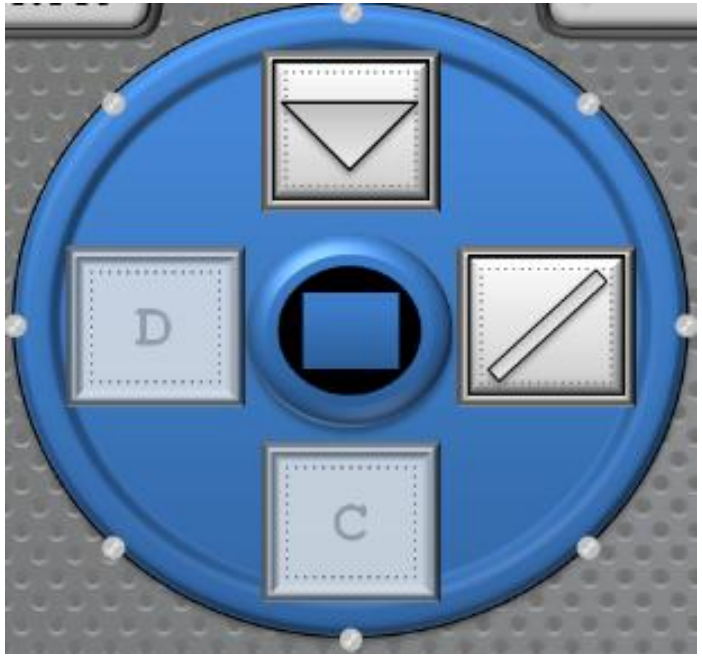

```
theta = Math.toRadians(180-theta);
double r = image.getHeight();
double x = r*Math.sin(theta) -(image.getWidth()/2)*Math.cos(theta) +pivotX;
double y = r*Math.cos(theta) +(image.getWidth()/2)*Math.sin(theta) +pivotY;
p1.setLocation(x,y);

x = r*Math.sin(theta) +(image.getWidth()/2)*Math.cos(theta) +pivotX;
y = r*Math.cos(theta) -(image.getWidth()/2)*Math.sin(theta) +pivotY;
p2.setLocation(x,y);

r = 0;
x = r*Math.sin(theta) +pivotX;
y = r*Math.cos(theta) +pivotY;
p3.setLocation(x,y);

walls.get(0).setLine(p1,p2);
walls.get(1).setLine(p2,p3);
walls.get(2).setLine(p3,p1);
```

You've Come to Understand how to Design a Class, and How Methods Apply to Instances



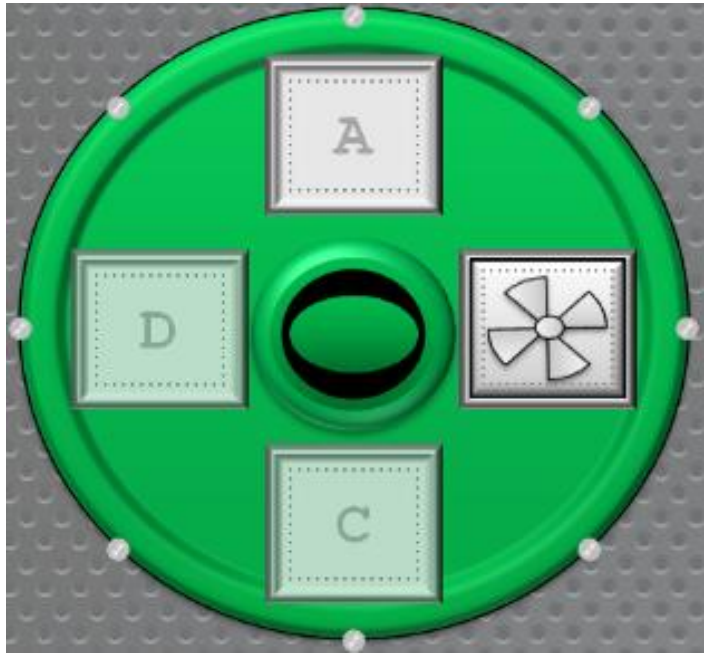
```
public class BlueBumper {
…

      public void methodA(){
           triangleWall();
      }
      public void methodB(){
           simpleWall();
      }
      public void methodC(){

      }
      public void methodD(){

      }
 }
```

And How Methods should be Inherited or Overridden



```
public class GreenBumper extends BlueBumper {
…

      public void methodB(){
           fan();
      }

 }
```

Benefits of Inheritance

- Inherited methods don't need to be re-written in sub classes.
  - They're written once in the super class.
  - For example, a Triangle Wall is found on Green Bumpers, but this behavior is never explicitly written in the `GreenBumper` class. It's written in the `BlueBumper` class.
- Remember `TriangleWall()` represents complex code.
  - Code is neater: You don't need to re-write all the complex stuff in multiple locations.
  - Maintainability: You make edits in one place. Changes apply to all inherited classes.

```
public class GreenBumper extends BlueBumper {
…
      public void methodB(){
            fan();
      }
 }
```

More Benefits of Inheritance

- You only need to explicitly write new or overriding methods in a sub class.
- This is why slots are labeled ABCD.
  - You need to know the name of the method you're overriding.
- For example, the implementation of `methodB()` differs between the `BlueBumper` super class and `GreenBumper` sub class.
  - `methodB()` is explicitly re-written in the `GreenBumper` class with a different implementation. The method overrides the Simple Wall implementation with a Fan.

```
public class GreenBumper extends BlueBumper {
…
      public void methodB(){
           fan();
      }
 }
```

You've Come Far

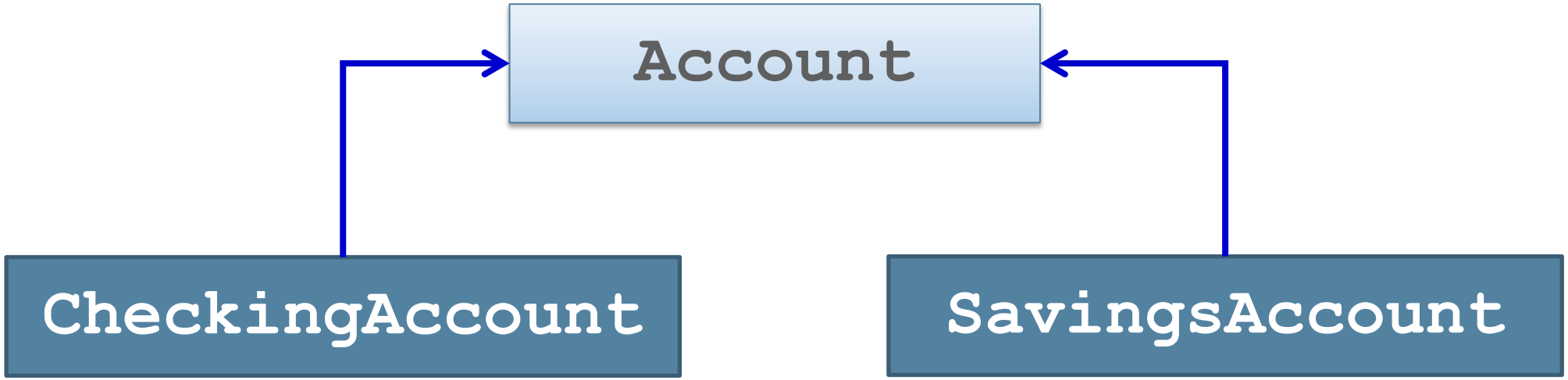
- Think about what problems you now have experience wrestling:
  - Planning and designing a class's fields and methods
  - Distributing methods wisely between classes
  - Utilizing Inheritance to achieve the desired class design
- You're now in a position where you can apply your understanding to work with real Java code, and better interpret the results of making changes.
  - The `SavingsAccount` and `CheckingAccount` classes of Lab 2 duplicated a lot of the same fields and methods. When this occurs between classes, it's an indication that it may be beneficial to create some sort of inheritance structure.
  - Completing this work involves the same thought processes you've already developed and strengthened by playing Java Puzzle Ball.

Lab 3: Finish the Inheritance Structure

- The Lab Instructions are available on the Lesson 3 page of the MOOC.
- As you work, consider…
  - Are there any fields and behaviors shared between classes which are better inherited from a super class?
  - Are there any fields or methods which are unique to the subclass?
  - Are there any inherited behaviors that need to be tweaked, or overridden?
- The remaining part of this lesson will give you tips.

The Inheritance Structure

- An inheritance structure is setup between the `Account` and `CheckingAccount` classes.
  - `Account` is the super class.
  - `CheckingAccount` is a sub class of `Account`.
  - Your goal is to also make `SavingsAccount` a sub class of `Account`.



The Abstract `Account` class

- Both fields and methods are inherited from a parent class.
  - The `Account` class outlines all the fields and methods that should be shared by any type of account.
- The `Account` class is abstract.
  - Classes are made abstract by placing the keyword `abstract` in the class declaration.
- Think of abstract classes as being incomplete. Because of this incompleteness…
  - They don't model any particular object.
  - Instances can't be created of an abstract class.
  - They're just a container for fields and methods to be inherited by other classes.

The `CheckingAccount` class

- This class contains:
  - One field which is unique to a Checking Account
  - Its own constructor
  - The `printDetails()` method, which overrides the method's inherited version.
- The remaining fields and methods are inherited from the `Account` class.
- You may also notice…
  - The constructor calls `super(o, b)`. This line of code executes the constructor of the super class. In other words, it's passing values to the super class's constructor and letting that method worry about assigning values to fields.
  - The `printDetails()` method calls `super.printDetails()`. This line of code executes the version of the method that's inherited from the super class.

The `SavingsAccount` class

- This class is unchanged from Lab 2.
- To keep code duplication to a minimum and increase maintainability, your goals are to:
  - Allow this class to inherit from the `Account` class.
  - Remove fields and methods that are better inherited rather than explicitly written in the class.
  - Leave the fields and methods that are unique to the class.
  - Provide any modifications to inherited methods whose implementation should be overridden.
- Study `CheckingAccount` to revise `SavingsAccount`.

Play with the `TestClass`

```
public class TestClass {
    public static void main(String[] args) {

    SavingsAccount savings1 = new
SavingsAccount("Duke", 100);
    SavingsAccount savings2 = new
SavingsAccount("Damien", 200);
    SavingsAccount savings3 = new
SavingsAccount("Jessica", 500);
    SavingsAccount savings4 = new
SavingsAccount("Herbert", 500);

    //Call methods on instances
    savings1.printDetails();
    savings2.printDetails();
    savings3.printDetails();
    savings4.printDetails();
    }
}
```

This shouldn't be necessary. It's better to print details automatically when an instance is created.

- The output can get long and complicated.
- You'll therefore notice a few things could be improved:
  - It's difficult to distinguish where a new account is created.
  - When an error occurs (like depositing a negative number), it's difficult to distinguish which account is affected.
  - Unnecessary blank lines may appear when printing an account's details.
- Edit your program to address these issues.

Lots More to Learn…

- Did you know you can have abstract methods too?
- Do you know what a Java Interface is?
  - You can inherit from both classes and interfaces.

- You won't need to know these concepts for this course.
  - But if you're curious, Oracle as other courses where you can learn more.


Java™
ORACLE®