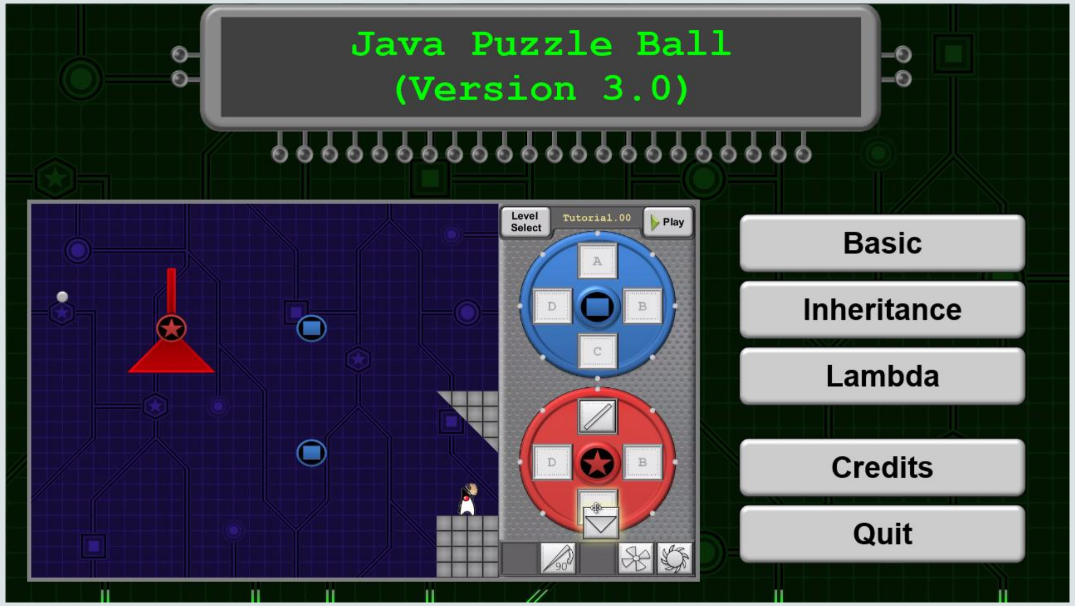
Java™

ORACLE®

Java Puzzle Ball

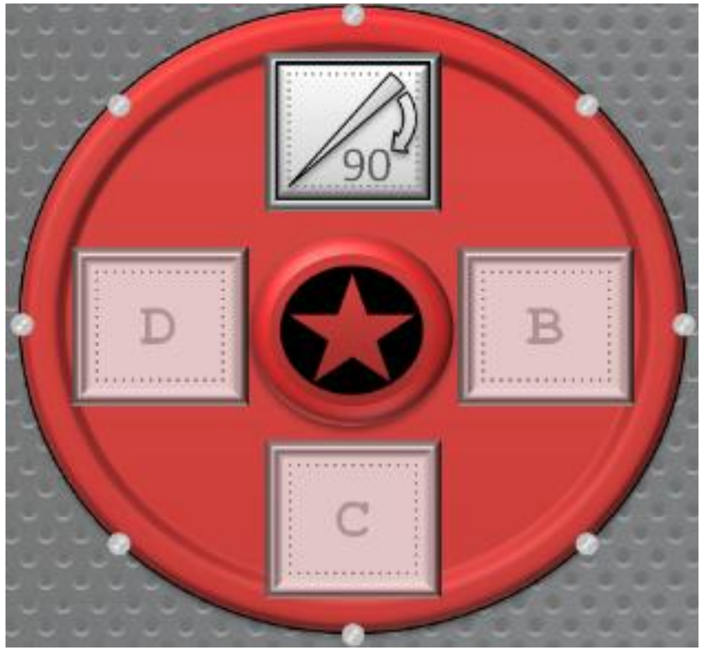
Nick Ristuccia

Lesson 2-3
Editing Java Code



You've seen Static Variables

- Red Bumpers start with an orientation of 0.

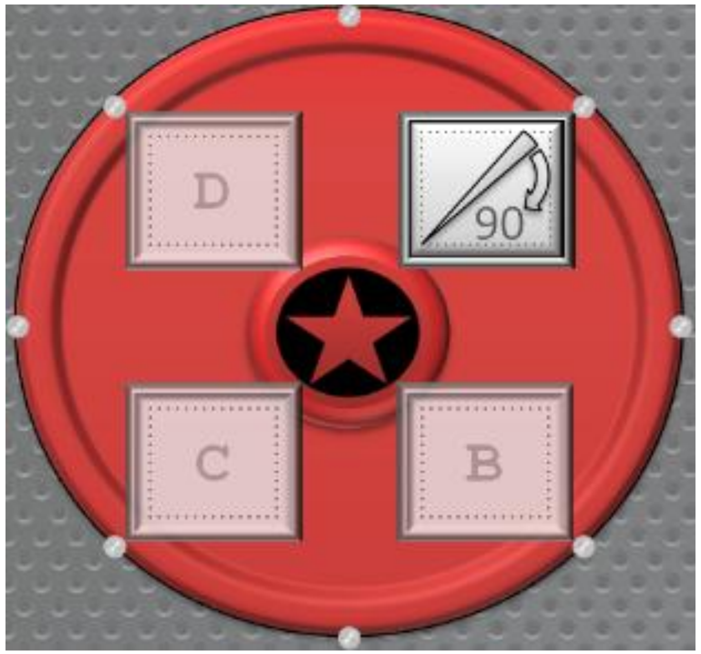


```
public class RedBumper {
    private static Color color = Color.RED;
    private static Shape shape = Shape.STAR;
    private static double orientation = 0;
    private double rotation;
    private int xPosition;
    private int yPosition;

…
    public void methodA(){
        rotation = rotation +90.0;
    }
…
```

You've seen Static Variables Change

- Rotating the Red Wheel changes the orientation.
  - This is like changing the field directly from 0 to 45.
  - The new value is applied to all instances of Red Bumpers.

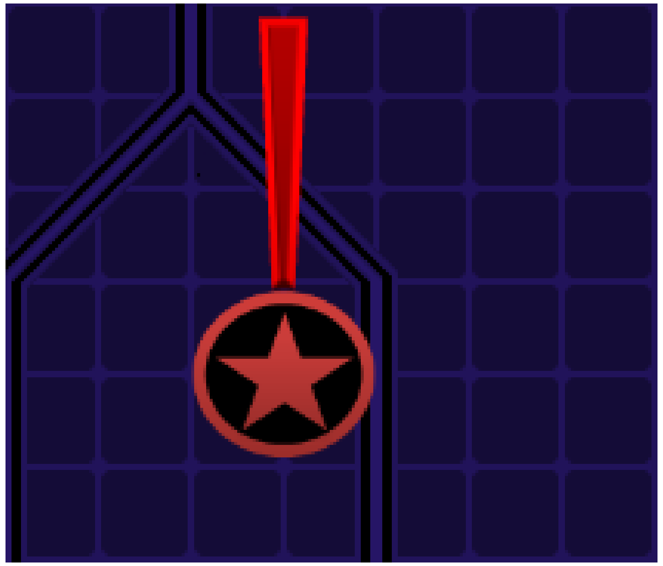


```
 public class RedBumper {
     private static Color color = Color.RED;
     private static Shape shape = Shape.STAR;
     private static double orientation = 45;
     private double rotation;
     private int xPosition;
     private int yPosition;


…
     public void methodA(){
         rotation = rotation +90.0;
     }
…
```

You've seen Instance Variables

- Individual Red Bumpers start with an additional rotation of 0.

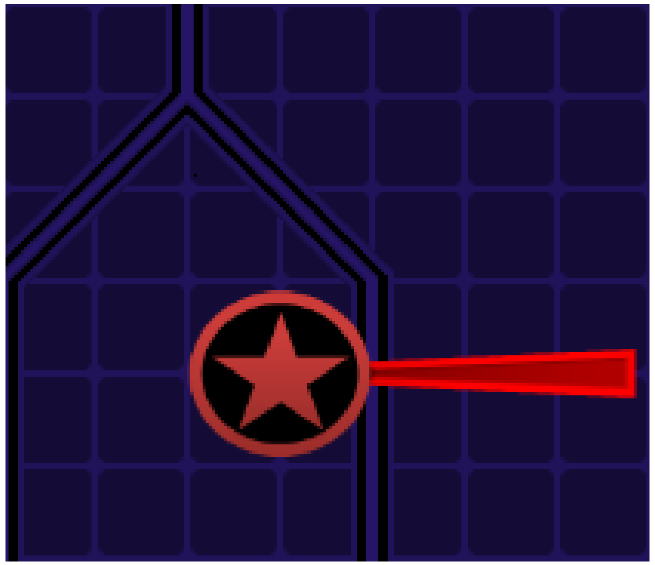


```
public class RedBumper {
     private static Color color = Color.RED;
     private static Shape shape = Shape.STAR;
     private static double orientation = 0;
     private double rotation;
     private int xPosition;
     private int yPosition;

…
     public void methodA(){
          rotation = rotation +90.0;
     }
…
```

You've seen Instance Variables Change

- An additional 90° of rotation is added to an individual bumper when struck.
  - This behavior of altering the `rotation` variable is implemented in `methodA()`.
  - When the method is called, the change applies to just one Red Bumper's rotation, and not every Red Bumper.



```
public class RedBumper {
     private static Color color = Color.RED;
     private static Shape shape = Shape.STAR;
     private static double orientation = 0;
     private double rotation;
     private int xPosition;
     private int yPosition;


…
     public void methodA(){
          rotation = rotation +90.0;
     }
…
```

You've Helped Write Code Where Instance Variables Change

- The `deposit()` method changes the value of the `balance` field.
  - The new value of `balance` is equal to the old `balance` + `x`.
  - When the method is called, a deposit is made into just one account, and not every account.

```
 public class SavingsAccount{
     private double balance;
…

     public void deposit(double x){
         balance = balance + x;
     }
…
}
```

Lab 2: Static and Instance Variables

- The Lab Instructions are available on the Lesson 2 page of the MOOC.
- As you work, consider…
  - Which properties should apply to the entire class?
  - Which properties should be different for each individual instance?
- The remaining part of this lesson will give you tips.

Which Fields should be Static?

- Carefully decide which fields should be static.
  - Add the `static` keyword to the appropriate fields
- The `accountNum` field is correctly marked non-static for you.
- The `nextAccountNum` field is correctly marked static for you.
  - You won't need to change these two fields.

```
public class SavingsAccount {
    private String accountType;
    private String accountOwner;
    private double balance;
    private double interestRate;
    private int accountNum;
    private static int nextAccountNum =0;
 …
 }
```

```
public class CheckingAccount {
    private String accountType;
    private String accountOwner;
    private double balance;
    private int accountNum;
    private static int nextAccountNum =0;
 …
 }
```

Explanation of `accountNum` and `nextAccountNum`

- These two fields are properly labeled non-static and static.
- Like bank accounts in real life, the accounts in this lab need a number.
  - `accountNum` represents the number assigned to an account.
  - `nextAccountNum` is a counter. It's necessary to keep track of numbers already issued to accounts, so that no two accounts share the same number. The field is static because the class only needs one counter.
  - The method `setAccountNumber()` assigns an account number, and then increments the counter by 1.

```
private void setAccountNumber(){
     accountNum = nextAccountNum;
     nextAccountNum++;
}
```

Lab 1 Had a Problem

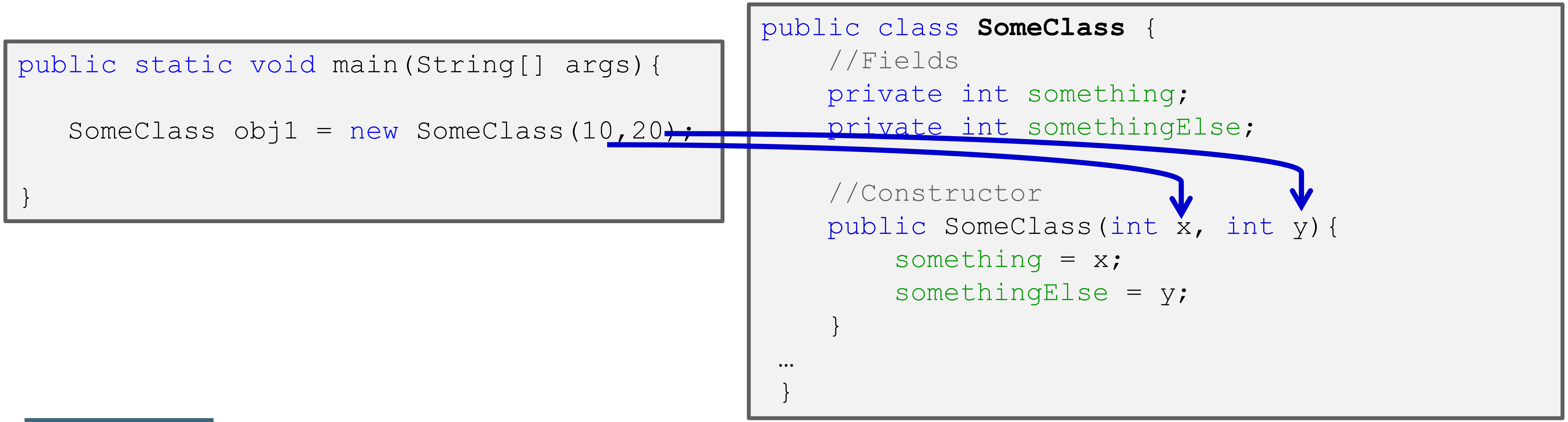
- All accounts owners are named Duke.
- But a good banking program should accommodate different names and other properties when an account is created.
- A special method called a **constructor** provides a great solution.

```
public class CheckingAccount {
    //Fields
    private String accountType;
    private String accountOwner;
    private double balance;
    private int accountNum;
    private static int nextAccountNum = 0;

    //Constructor
    public CheckingAccount(String t, String o, double b){
        accountType = t;
        accountOwner = o;
        balance = b;
        setAccountNumber();
    }
…
 }
```

Constructors set Instance Variable Values

- When you create a new instance of an object, you're actually calling its constructor.
  - And passing values to the constructor, which are used to set the initial values of fields.



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

   SomeClass obj1 = new SomeClass(10,20);

}
```

```
public class SomeClass {
    //Fields
    private int something;
    private int somethingElse;

    //Constructor
    public SomeClass(int x, int y){
        something = x;
        somethingElse = y;
    }
 …
 }
```

Constructors (Usually) Don't Set Static Variables

- In the Lab 2 start state, most fields treat their variable like an instance variable.
- If you believe a field should instead be static, remove the variable from the constructor and set variable's initial value where the field is declared.

Example of Modifying a Constructor

Before

```
public class SomeClass {
    //Fields
    private int something;
    private int somethingElse;

    //Constructor
    public SomeClass(int x, int y){
        something = x;
        somethingElse = y;
    }
 …
 }
```

After

```
public class SomeClass {
    //Fields
    private int something;
    private static int somethingElse = 0;

    //Constructor
    public SomeClass(int x){
        something = x;

    }
 …
 }
```

Play with the `TestClass`

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

    //Create new instances
    SavingsAccount savings1 = new
SavingsAccount("Savings Account", "Duke", 100, 0.02);
    //Call methods on instance


    //Create new instances
    CheckingAccount checking1 = new
CheckingAccount("Checking Account", "Duke", 0);
    CheckingAccount checking2 = new
CheckingAccount("Checking Account", "Mrs.Duke",
500000);
    //Call methods on instances
    checking1.printDetails();
    checking2.printDetails();
    }
}
```

- Play with the `main` method to test Savings and Checking Account instances.
  - Create several instances of each account type.
  - Observe the values of instances.
  - If two people have the same name, will their accounts at least have different balances?
- Based on your edits, you may need to remove some values previously passed to the constructors.

Lots More to Learn…

- Did you know there are Static Methods too?
- Did you know you can have many methods, all with the same name?

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

ORACLE® ACADEMY

ORACLE® UNIVERSITY

ORACLE® LEARNING LIBRARY

Java™

ORACLE®