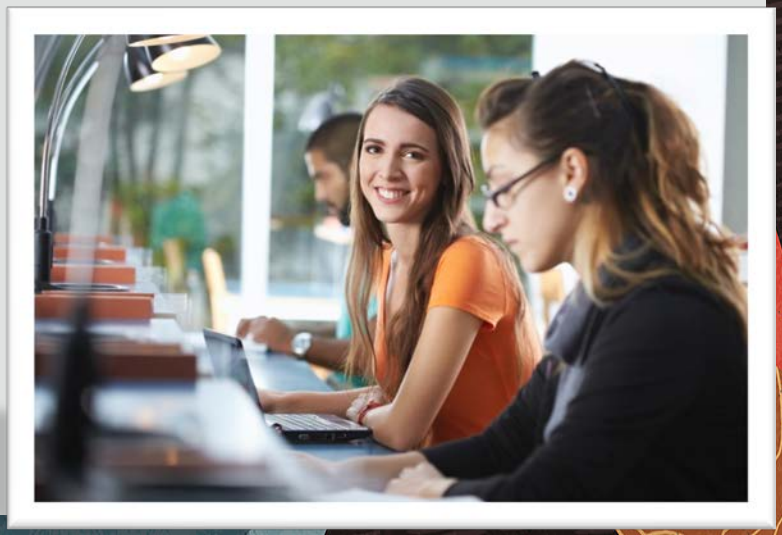

ORACLE
Academy

Java Foundations

7-6
Static Variables and Methods



ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - Describe a static variable and demonstrate its use within a program
  - Describe a static method and demonstrate its use within a program
  - Understand how to use the final keyword with static variables

Review of Object References

- An object must be instantiated before its fields and methods can be accessed
- Instantiation provides us with an object reference
- An object reference is used to access an object's fields and methods

```
Prisoner p01 = new Prisoner()
p01.name                    //Accessing a field
p01.display()               //Calling a method
```

The Math Class Is Different

- It would be tedious to create a new Math object every time we wanted to do a little math
- Thankfully, we never need to instantiate a Math object
- Math fields and methods are accessed by directly referencing the Math class
- These are known as static variables and static methods

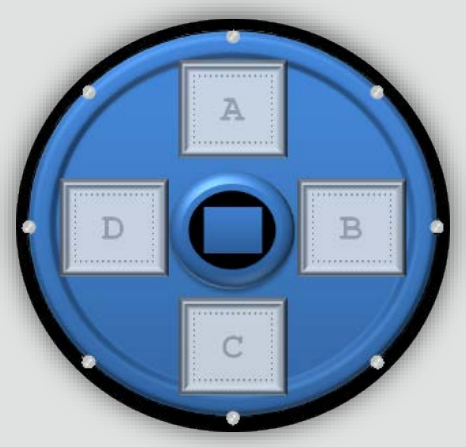
```
                              //Nothing instantiated
Math.PI                       //Accessing a static field
Math.sin(0)                   //Calling a static method
```

What Does This Mean?

- Why are these two facts important?
  - An object reference is used to access an object's fields and methods
  - Static fields and methods are accessed by directly referencing the class
- There's more to it than just the convenience of not having to instantiate an object
- The next exercise lets you explore a use-case for static data
  - Then we'll debrief you on what you may have noticed

Exercise 1

- Play Basic Puzzles 8 through 11
  - https://objectstorage.uk-london-1.oraclecloud.com/n/lrvrlgaqj8dd/b/Games/o/JavaPuzzleBall/index.html
  - Consider the following:
  - What happens when you rotate the BlueWheel?
  - How else can you affect the rotation of bumpers?

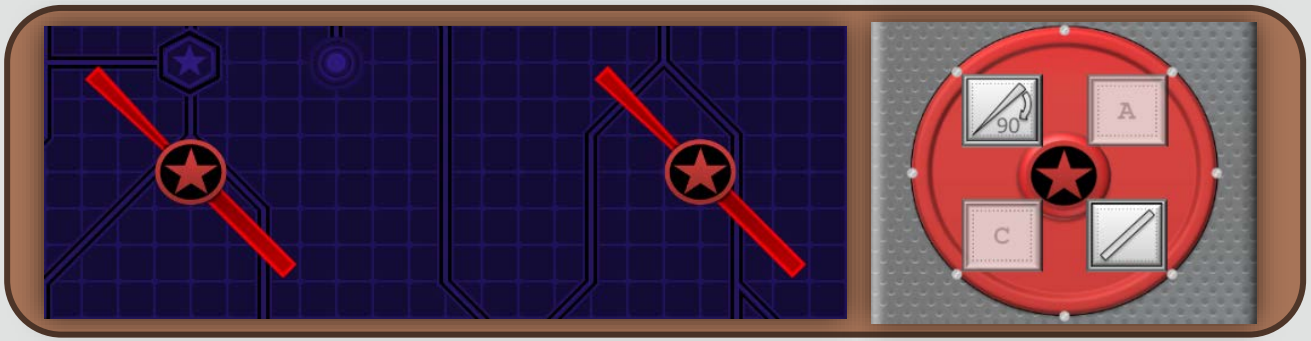


Java Puzzle Ball Debriefing

- What happens when you rotate the BlueWheel?
  - The orientation of all BlueBumpers change
  - All BlueBumpers share the orientation property
  - Orientation can be represented by a static variable
- How else can you affect the rotation of bumpers?
  - After the ball strikes a rotation wall, the rotation of an individual bumper changes
  - Rotation can be represented by an instance variable

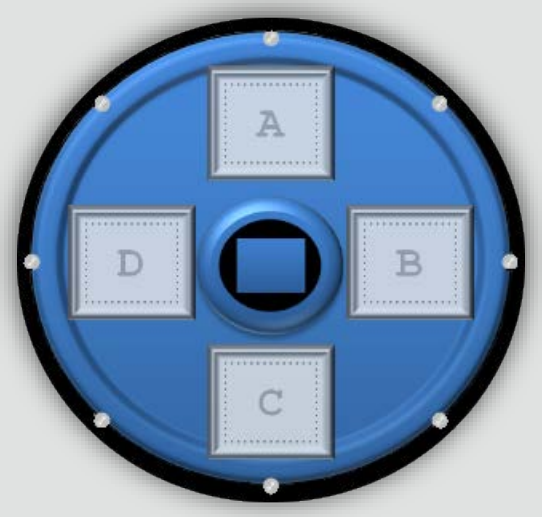
Static Variable: Orientation

- This static variable is shared by all instances
- Static variables belong to the class, not to any individual instance
- Therefore, a static variable needs to be changed only once for every instance to be affected
- In Basic Puzzle 11, rotating the RedWheel changes the orientation of all RedBumper objects



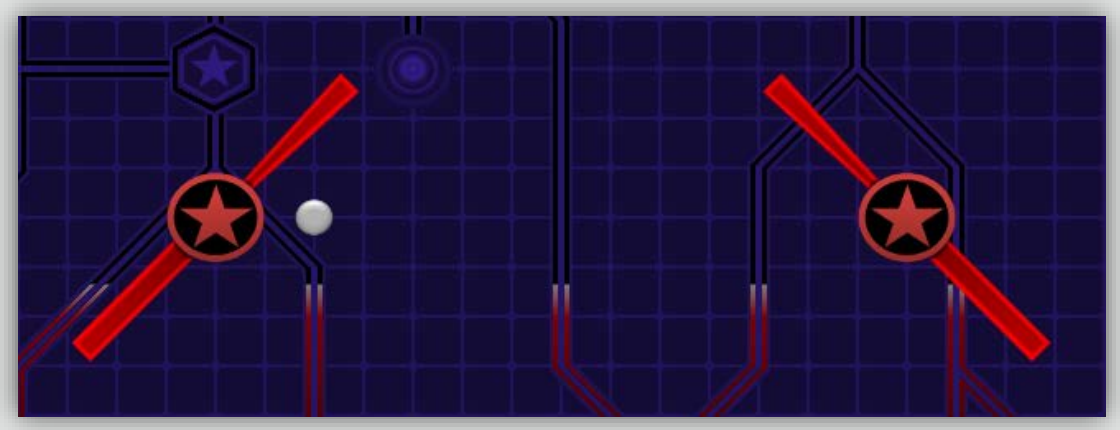
Static Variables with No Instances

- Static variables can be accessed, even if no objects have been instantiated
- In Basic Puzzle 11, the BlueWheel can be rotated to change the orientation property of all BlueBumpers
  - There just aren't any BlueBumpers to show the effects of this change



Instance Variables: Rotation

- Unique instance variables exist for every instance of an object
- Therefore, instance variables need to be changed for each individual object
- In Basic Puzzle 11, an individual RedBumper's rotation changes after being struck by the ball



When Should a Field Be Static?

- Here are a few points to consider:
  - Will the value of this field be different for each individual object? Or will it be the same for all objects?
  - Does the field describe the class more than it describes any individual object?
  - Do you find yourself repeating the same value throughout the class?
  - Is this value a constant that will be used in calculations?
  - Will this value need to be accessed before any objects are instantiated?

Creating Static Variables

- A variable becomes static when its declaration includes the static keyword
- Initialize static variables as they're declared
  - Otherwise, repeated constructor calls could "initialize" the same static variable many times

```
public class RedBumper{
   //Fields
   public static int orientation = 45;  //Static variable
   public int rotation;                 //Instance variable

   //Constructor
   public RedBumper(int rotation){
      this.rotation = rotation;
   }//end constructor
}//end class RedBumper
```

Accessing Static Variables in Their Class

- Even if static variables aren't initialized in the constructor, they can still be accessed
- Like any other variable, static variables are accessible within their class

```
public class RedBumper{
   //Fields
   public static int orientation = 45;  //Static variable
   public int rotation;                 //Instance variable
   …
   //Methods
   public void display(){
      System.out.println(orientation); //Access static var
      System.out.println(rotation);    //Access instance var
   }//end method display
}//end class RedBumper
```

Accessing Static Variables Elsewhere

- Static variables can appear in constructors, methods, or outside their class
- Calling static variables outside their class relies on referencing the class's name rather than a specific reference variable

```
public class TestClass {
   public static void main(String[] args){
      int x;
      x = RedBumper.orientation;   //Access static variable

      RedBumper rb01 = new RedBumper(90); //Instance
      int y;
      y = rb01.rotation;          //Access instance variable
   }//end method main
}//end class TestClass
```

Exercise 2

- Continue editing the `PrisonTest` project
  - A version of this program is provided for you in the files `PrisonTest_Student_7_6.java`, `Prisoner_Student_7_6.java`, and `Cell_Student_7_6.java`
- Modify the `Prisoner` class:
  - Include a static integer prisonerCount field
  - This field counts the total number of prisoners instantiated
  - Initialize this field to 0
  - Increase this field every time a prisoner is instantiated
  - Include an integer bookingNumber field
  - This field is initialized with the current value of prisonerCount
  - Print the bookingNumber and prisonerCount as part of the display() method
- Instantiate a few prisoners and display their info.

You don't need to write getters for this exercise

Introducing Static Methods

- You may have noticed from the previous exercise:
  - The display() method can access a static variable
  - Static variables are accessible from nonstatic methods
- Most methods you've written in this course (excluding the main method) are considered instance methods
  - Instance methods are nonstatic methods
- Methods can also be made static

When Should a Method Be Static?

- Here are a few points to consider:
  - Will the method read or modify static fields?
  - Will the method not read or modify the fields of any particular object?
  - Will the method need to be called before any objects are instantiated?
- Static methods are for dealing with static data
  - Static variables are accessible from static methods

Creating Static Methods

- A method becomes static when its declaration includes the static keyword

```
public class Prisoner{
   //Fields
   private static int prisonerCount = 0;  //Static variable
   private int bookingNumber;             //Instance variable

   //Methods
   public static void displayPrisonerCount(){ //Static method
      System.out.println(prisonerCount);
   }//end method displayPrisonerCount
}//end class Prisoner
```

Calling Static Methods in Their Class

- Like any other method, static methods are callable within their class
- Static or instance methods may call a static method

```
public class Prisoner{
   private static int prisonerCount = 0;  //Static variable
   private int bookingNumber;             //Instance variable

   public static void displayPrisonerCount(){ //Static method
      System.out.println(prisonerCount);
   }//end method displayPrisonerCount
   public void callAnotherMethod(){         //Instance method
      displayPrisonerCount();
   }//end method callAnotherMethod
}//end class Prisoner
```

Calling Static Methods Elsewhere

- Static methods can be called from constructors, other methods, or outside their class
- Calling static methods outside their class relies on referencing the class's name rather than a specific reference variable

```
public class TestClass {
   public static void main(String[] args){
      Prisoner.displayPrisonerCount();  //Call static method

      Cell cA1 = new Cell("A1", false, 1234);
      Prisoner bubba = new Prisoner("Bubba", 2.08, 4, cA1);
      bubba.display();                  //Call instance method
   }//end method main
}//end class TestClass
```

Exercise 3

- Continue editing the `PrisonTest` project
- Modify the `Prisoner` class:
  - Encapsulate the prisonerCount field. Make this field private and create a static getter method
  - Try making the display method static
  - What are your IDE's complaints?
- From the main method:
  - Call the getter method that you just created and print the returned value

Why Did your IDE Complain?

- Static fields and static methods can be called without instantiating an object
- But instance variables must be associated with a specific instance
- A paradox is created if a static method tries to access information about an instance before it's created
- Therefore, Java doesn't allow static methods to contain instance variables or instance methods

```
public static void display(){
   System.out.println(prisonerCount);
   System.out.println(bookingNumber);
}//end method display
```

Writing static final Fields

- You're encouraged to make static variables final
  - But the reasons are beyond the scope of this course
- Remember, the names of final variables …
  - Are capitalized by convention
  - Use an underscore (_) to separate words

```
public class Prisoner{
   //Fields
    …
   private int bookingNumber;
   private static int prisonerCount = 0;
   public static final int MAX_PRISONER_COUNT = 100;
}//end class Prisoner
```

Making static final Primitive Fields public

- Encapsulation prevents variables from being manipulated in an undesirable way
- But there's no risk of public static final primitives being tampered with because it's impossible for their values to change
- This is useful for constants such as π, e, or other values constantly used in calculations
- These variables are called directly instead of through getters

```
System.out.println(Math.PI);
System.out.println(Math.E);
```

Summary

- In this lesson, you should have learned how to:
  - Describe a static variable and demonstrate its use within a program
  - Describe a static method and demonstrate its use within a program
  - Understand how to use the final keyword with static variables


ORACLE
Academy