



Java Foundations

7-1

Creating a Class



ORACLE ACADEMY

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

2

Objectives

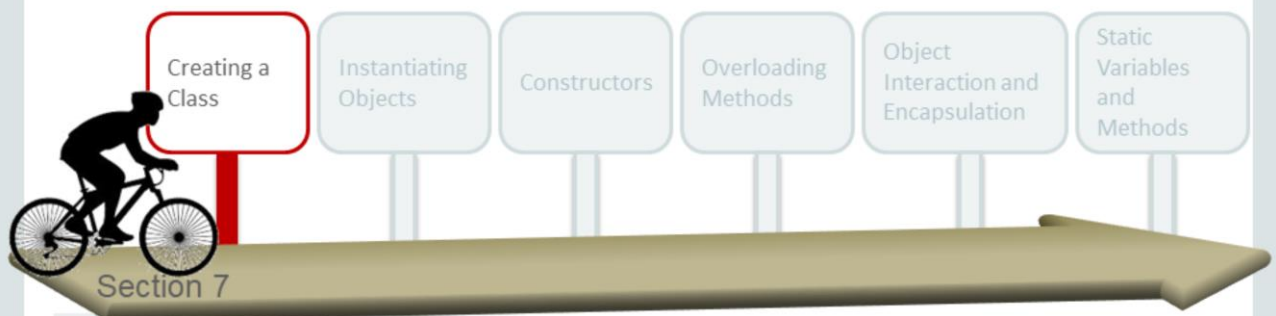
This lesson covers the following objectives:

- Create a Java test/main class
- Create a Java class in NetBeans
- Use conditionals in methods
- Translate specifications or a description into fields and behaviors



Topics

- Object-Oriented Thinking
- Object-Oriented Programming
- Scope of Variables
- Creating a Class from Specifications



Object-Oriented Concepts

- We've been experimenting with conditional statements and loops for a while
- Now would be a good time to review object-oriented programming concepts and its benefits
- The rest of this section describes object-oriented programming in greater detail

Exercise 1



- Play **Basic Puzzles 6 and 7**.
 - Your Goal: Design a solution that deflects the ball to Duke.
- Consider the following:
 - What happens when you put an icon on the blue wheel?

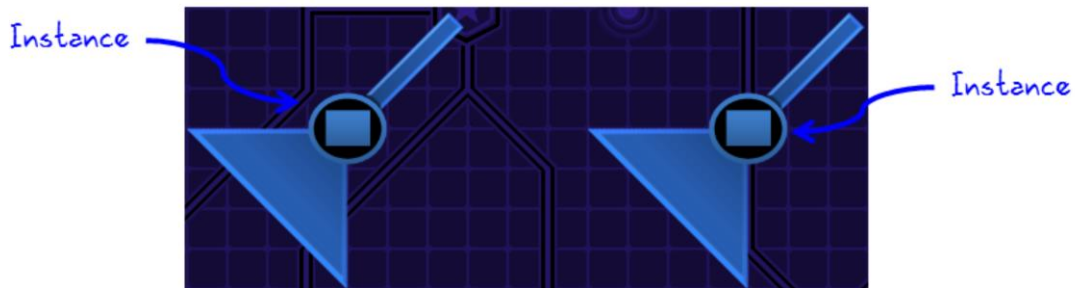


Java Puzzle Ball Debrief



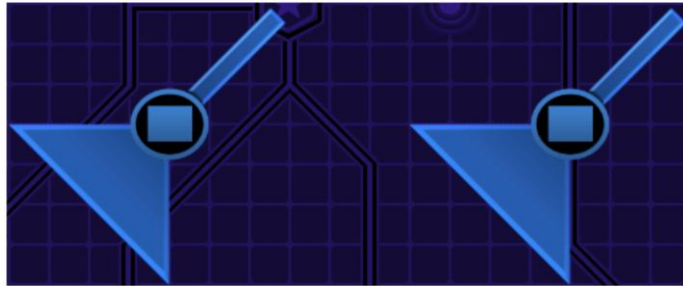
What happens when you put icons within a blue wheel?

- A wall appears on every **instance** of a blue bumper **object**.
- Walls give bumpers **behaviors** that deflect and interact with the ball.
- All blue bumper instances share these same behaviors.



A blue bumper is an object, and every instance of these objects share the same behavior for interacting with the ball. These behaviors may include deflection via triangle or the simple wall.

Describing a Blue Bumper



(Fields)

- Properties:
 - Color
 - Shape
 - x-position
 - x-position

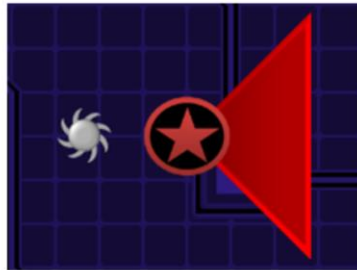
(Methods)

- Behaviors:
 - Make ping sound
 - Flash
 - Deflect ball
 - Get destroyed

Conditional Logic and Loops in Classes

- Conditionals and loops can play an important role in the methods you write for a class.
- The main method was a convenient place to experiment and learn conditional logic and loops.
- But remember ...
 - The main method is meant to be a driver class.
 - Your entire program shouldn't be written in the main method.

What If the Ball Collides with a Bumper?



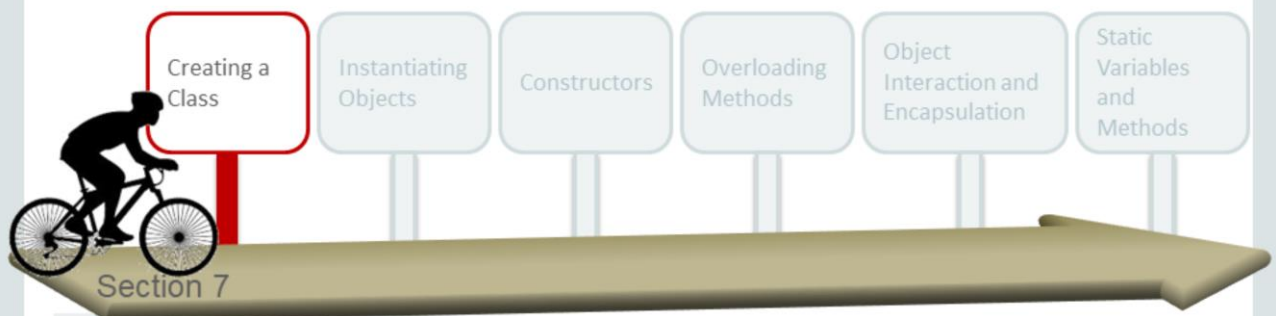
A method with the following logic is called:

```
public void onCollisionWithBall(Ball ball){  
    if(ball.isBlade == true){           //Ball is blade  
        getDestroyed();  
    }  
    else{                               //Ball is not blade  
        deflectBall();  
    }  
}
```

The `RedBumper` has a method for handling collisions. When this method is called, it checks to see if the ball is a blade. `isBlade` is a boolean property belonging to the `Ball` class. If the ball is a blade, the bumper is destroyed. Otherwise, the ball is deflected. Getting destroyed and deflecting the ball are behaviors of a `Bumper`. This is made possible with the `getDestroyed()` and `deflectBall()` methods.

Topics

- Object-Oriented Thinking
- Object-Oriented Programming
- Scope of Variables
- Creating a Class from Specifications



Modeling a Savings Account

- You could model one savings account like this:

```
public class SavingsAccount{  
  
    public static void main(String args[]){  
  
        int balance = 1000;  
        String name = "Damien";  
    }  
}
```

- And two accounts like this:

```
int balance1 = 1000;  
String name1 = "Damien";  
  
int balance2 = 2000;  
String name2 = "Bill";           //Copy, Paste, Rename
```

Modeling Many Accounts

- How would you model 1000 accounts?

```
...  
    //You think ...  
    //Do I really have to copy and paste 1000 times?
```

- How would you add a parameter for each account?

```
...  
    //You think ...  
    //There has to be a better way!
```

- There is a better way: Use a class.
 - And not the main method.

How to Structure a Class

- Code should fit this format:

```
1 public class SavingsAccount {  
2  
3     Properties  
4  
5  
6     Behaviors  
7  
8  
9 }
```

How to Structure a Class

- Code should fit this format:

```
1 public class SavingsAccount {  
2     public double balance;  
3     public double interestRate = 0.01;  
4     public String name;  
5  
6     public void displayCustomer() {  
7         System.out.println("Customer: "+name);  
8     }  
9 }
```


- With one simple line of code(line 3), all 1000 accounts have an interest rate.
 - And we can change the rate at any time for any account.

The Main Method as a Driver Class

- Place the main method in a test class.
 - The main method is often used for instantiation.

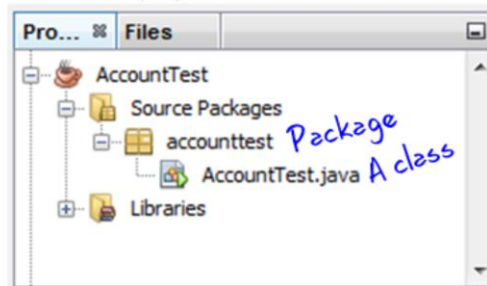
```
1 public class AccountTest {  
2     public static void main(String[] args){  
3  
4         SavingsAccount sa0001 = new SavingsAccount();  
5         sa0001.balance = 1000;  
6         sa0001.name = "Damien";  
7         sa0001.interestRate = 0.02;  
8  
9         SavingsAccount sa0002 = new SavingsAccount();  
10        sa0002.balance = 2000;  
11        sa0002.name = "Bill";  
12 }
```


How to Create a Java Project

1. In NetBeans, click **New Project** ().
2. For Category, select **Java**. For Project, select **JavaFX Application**. Click **Next**.
3. Name your project.
4. Click **Browse** and select the location where you want to store the project.
5. Select the check box to automatically create a main method.
6. Click **Finish**.

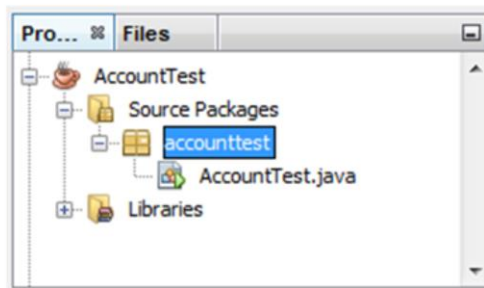
You've Created a Project with a Test Class

- The class contains a main method.
- If you expand the project directory, you'll notice ...
 - The `accounttest` package
 - Your test class (`AccountTest.java`)
- Think of a package like a folder.
 - You're able to add many `.java` files to this folder



How to Add a Class to a Project

1. Right-click the package where you want to create the class.
2. Choose: **New >> Java Class**
3. Name your class.
 - Classes always start with a **CAPITAL LETTER**.
 - Every subsequent word in the name is capitalized.
 - This is called **Camel Casing**.
4. Click **Finish**.



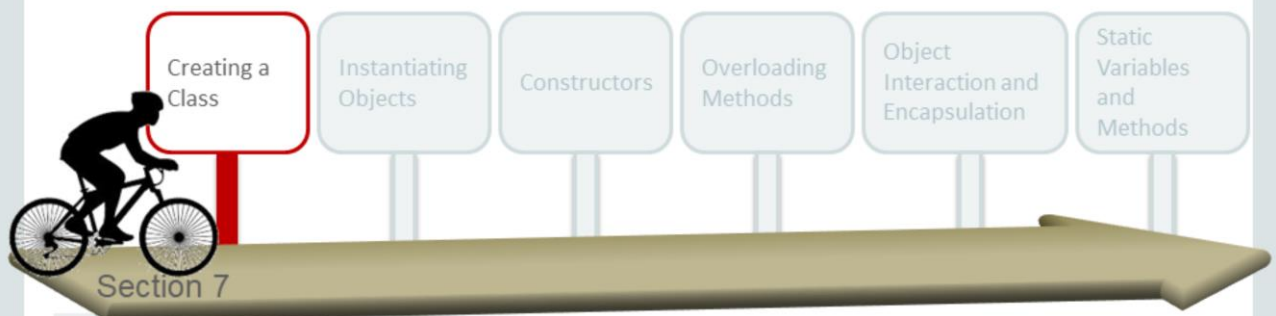


Exercise 2

- Create a new Java project.
- Create an `AccountTest` class with a main method.
- Create a `CheckingAccount` class.
 - Include fields for `balance` and `name`.
- Instantiate a `CheckingAccount` object from the main method.
 - Assign values for this object's `balance` and `name` fields.

Topics

- Object-Oriented Thinking
- Object-Oriented Programming
- Scope of Variables
- Creating a Class from Specifications



Variable Scope

- Fields are accessible anywhere in a class.
 - This includes within methods.

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;  
  
    public void displayCustomer() {  
        System.out.println("Customer: "+name);  
        System.out.println("Balance: " +balance);  
        System.out.println("Rate: "    +rate);  
    }  
}
```

Variable Scope

- Variables created within a method cannot be accessed outside that method.
 - This includes methods parameters.

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;  
}
```

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

Scope of x



```
public void badMethod() {  
    System.out.println(x);  
}
```

Not scope of x

Accessing Fields and Methods from Another Class

1. Create an instance.
2. Use the dot operator (.)

```
public class AccountTest {  
    public static void main(String[] args){  
  
        1) SavingsAccount sa0001 = new SavingsAccount();  
        2) { sa0001.name = "Damien";  
            sa0001.deposit(1000);  
        }  
    }  
}
```

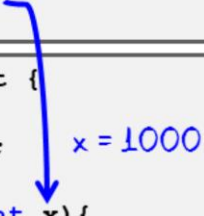
```
public class SavingsAccount {  
    public String name;  
    public double balance;  
  
    public void deposit(int x){  
        balance += x;  
    }  
}
```


Passing Values to Methods

- 1000 is passed to the `deposit()` method.
- The value of `x` becomes 1000.

```
public class AccountTest {  
    public static void main(String[] args){  
  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.name = "Damien";  
        sa0001.deposit(1000);  
    }  
}
```

```
public class SavingsAccount {  
    public String name;  
    public double balance;  
  
    public void deposit(int x){  
        balance += x;  
    }  
}
```





Exercise 3

- Continue editing the `AccountTest` project.
- Write a `withdraw()` method for checking accounts that ...
 - Accepts a double argument for the amount to be withdrawn.
 - Prints a warning if the balance is too low to make the withdrawal.
 - Prints a warning if the withdrawal argument is negative.
 - If there are no warnings, the withdrawal amount is subtracted from the balance. Print the new balance.
- Test this method with the instance from Exercise 2.

What if I Need a Value from a Method?

- Variables are restricted by their scope.
- But it's still possible to get the **value** of these variables out of a method.

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;
```

```
    public void calcInterest(){  
        double interest = balance*interestRate/12;  
    }  
}
```

Scope of
interest

Returning Values from Methods

- If you want to get a value from a method ...
 - Write a return statement.
 - Change the method type from `void` to the type that you want returned.

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;  
  
    //This method has a double return type  
    public double calcInterest(){  
        double interest = balance*interestRate/12;  
        return interest;  
    }  
}
```

Returning Values: Example

- When `getInterest()` returns a value ...

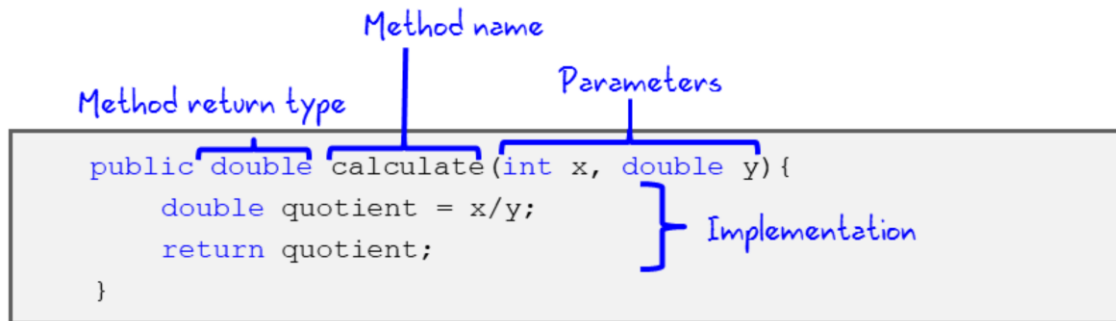
```
public class AccountTest {  
    public static void main(String[] args){  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.balance = 1000;  
        sa0001.balance += sa0001.calcInterest();  
    }  
}
```

- It's the equivalent of writing ...

```
public class AccountTest {  
    public static void main(String[] args){  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.balance = 1000;  
        sa0001.balance += 0.83;  
    }  
}
```

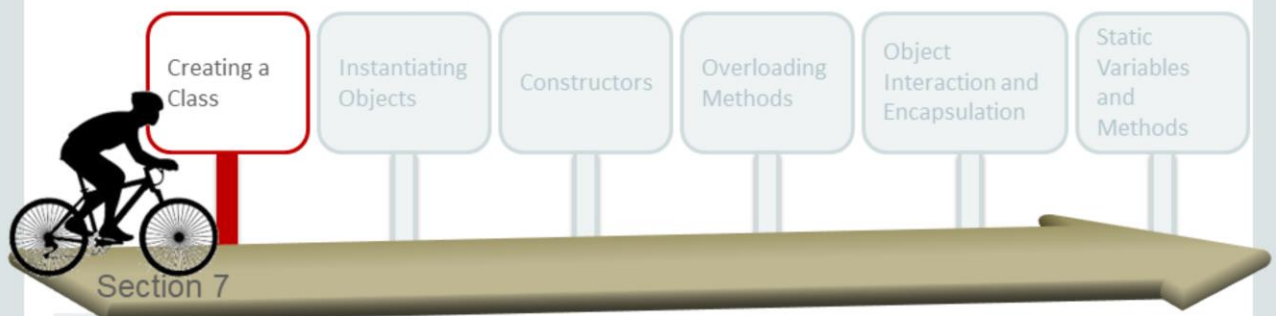
- But it's better and more flexible because the value is calculated instead of hard-coded.

Summary About Methods



Topics

- Object-Oriented Thinking
- Object-Oriented Programming
- Scope of Variables
- Creating a Class from Specifications



Limiting the Main Method

- The main method should be as small as possible.
- The example below isn't very good because ...
 - Increasing an account's balance based on interest is a typical behavior of accounts.
 - The code for this behavior should instead be written as a method within the `SavingsAccount` class.
 - It's also dangerous to have an account program where the balance field can be freely manipulated.

```
public class AccountTest {  
    public static void main(String[] args){  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.balance = 1000;  
        sa0001.balance += sa0001.calcInterest();  
    }  
}
```


The Rest of This Section

- We'll learn how to avoid these problematic scenarios when developing a class.
- But for this lesson, just focus on understanding how to:
 - Interpret a description or specification
 - Break it into properties and behaviors
 - Translate those properties and behaviors into fields and methods

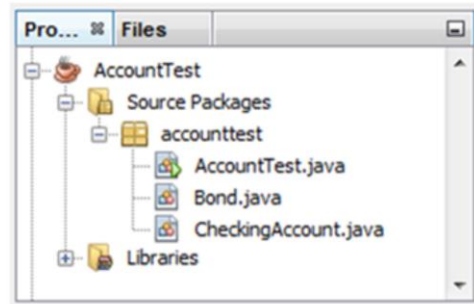


Exercise 4

- Continue editing the `AccountTest` project.
- Create a new class according to the description. Be sure to instantiate this class and test its methods.

Create a Savings Bond. A person may purchase a bond for any term between 1 and 60 months. A bond earns interest every month until its term matures (0 months remaining). The term and interest rate are set at the same time. The bond's interest rate is based on its term according to the following tier system:

- 0–11 months: 0.5%
- 12–23 months: 1.0%
- 24–35 months: 1.5%
- 36–47 months: 2.0%
- 48–60 months: 2.5%



Describing a Savings Bond



- Properties:

- Name
- Balance
- Term
- Months Remaining
- Interest Rate

- Behaviors:

- Set Interest Rate Based on Term
- Earn Interest
- Mature (0 months remaining)

You may have come up with these fields and behaviors.

Translating to Java Code: Part 1

- Your `Bond` class may have represented fields like this:

```
public class Bond{  
    public String name;  
    public double balance, rate;  
    public int term, monthsRemaining;  
    ...  
}
```

You may have written your program like this. Translating the fields into data types is an easy exercise.

Translating to Java Code: Part 2

- And include the following methods:

```
public void setTermAndRate(int t){
    if(t>=0 && t<12)
        rate = 0.005;
    else if(t>=12 && t<24)
        rate = 0.010;
    else if(t>=24 && t<36)
        rate = 0.015;
    else if(t>=36 && t<48)
        rate = 0.020;
    else if(t>=48 && t<=60)
        rate = 0.025;
    else{
        System.out.println("Invalid Term");
        t = 0;
    }
    term = t;
    monthsRemaining = t;
}
```

The rate depends on the term. An if/else construct can be used to check the value of the term and assign the correct rate based on that term. It's also useful to check if the term is invalid. If this happens, the term is set to 0 to prevent goofy account behavior.

Translating to Java Code: Part 3

```
...  
    public void earnInterest(){  
        if(monthsRemaining > 0){  
            balance += balance*rate/12;  
            monthsRemaining--;  
            System.out.println("Balance: $" +balance);  
            System.out.println("Rate: " +rate);  
            System.out.println("Months Remaining:"  
                               +monthsRemaining);  
        }  
        else{  
            System.out.println("Bond Matured");  
        }  
    }  
}
```

Earning interest is the last behavior you need to translate into a method. An if/else statement is useful here as well. If there are months remaining in the term of the CD, interest is added, and one fewer month is remaining to earn interest. If there aren't any months remaining, the CD is mature and cannot earn interest.

Summary

In this lesson, you should have learned how to:

- Create a Java test/main class
- Create a Java class in NetBeans
- Use conditionals in methods
- Translate specifications or a description into fields and behaviors



