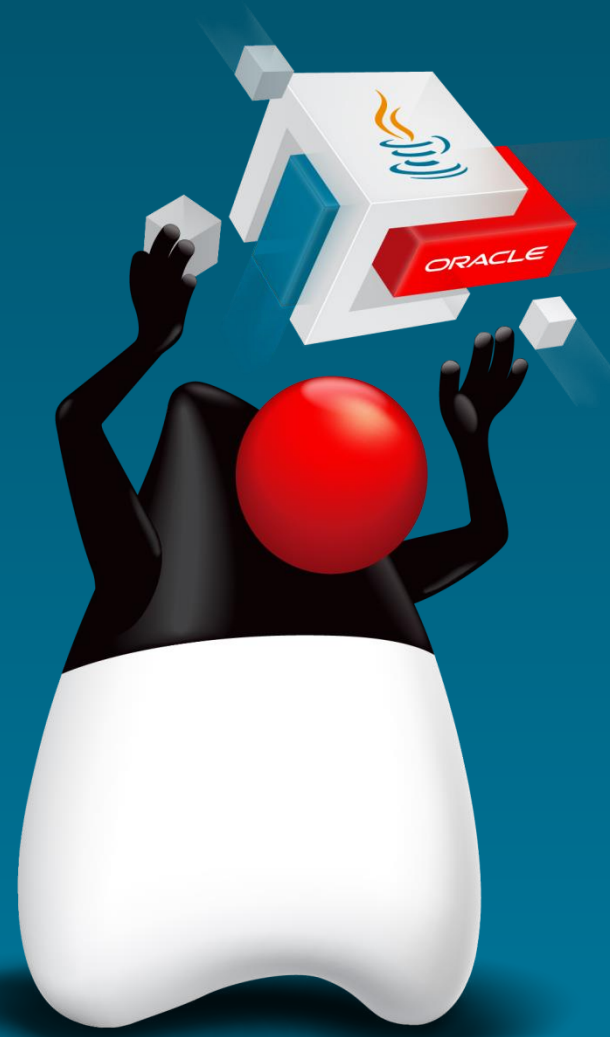


Creating and Using Methods

8



ORACLE®



Objectives

After completing this lesson, you should be able to:

- Add an argument to a method
- Instantiate a class and call a method
- Overload a method
- Work with static methods and variables
- Convert data values using `Integer`, `Double`, and `Boolean` object types



Topics

- Using methods and constructors
- Method arguments and return values
- Using static methods and variables
- Understanding how arguments are passed to a method
- Overloading a method



Basic Form of a Method

The void keyword indicates that the method does not return a value.

Empty parentheses indicate that no arguments are passed to the method.

```
1 public void display () {  
2     System.out.println("Shirt description:" + description);  
3     System.out.println("Color Code: " + colorCode);  
4     System.out.println("Shirt price: " + price);  
5 } // end of display method
```

Calling a Method from a Different Class

```
1 public class ShoppingCart {  
2     public static void main (String[] args) {  
3         Shirt myShirt = new Shirt();  
4         myShirt.display();  
5     }  
6 }
```

Reference variable

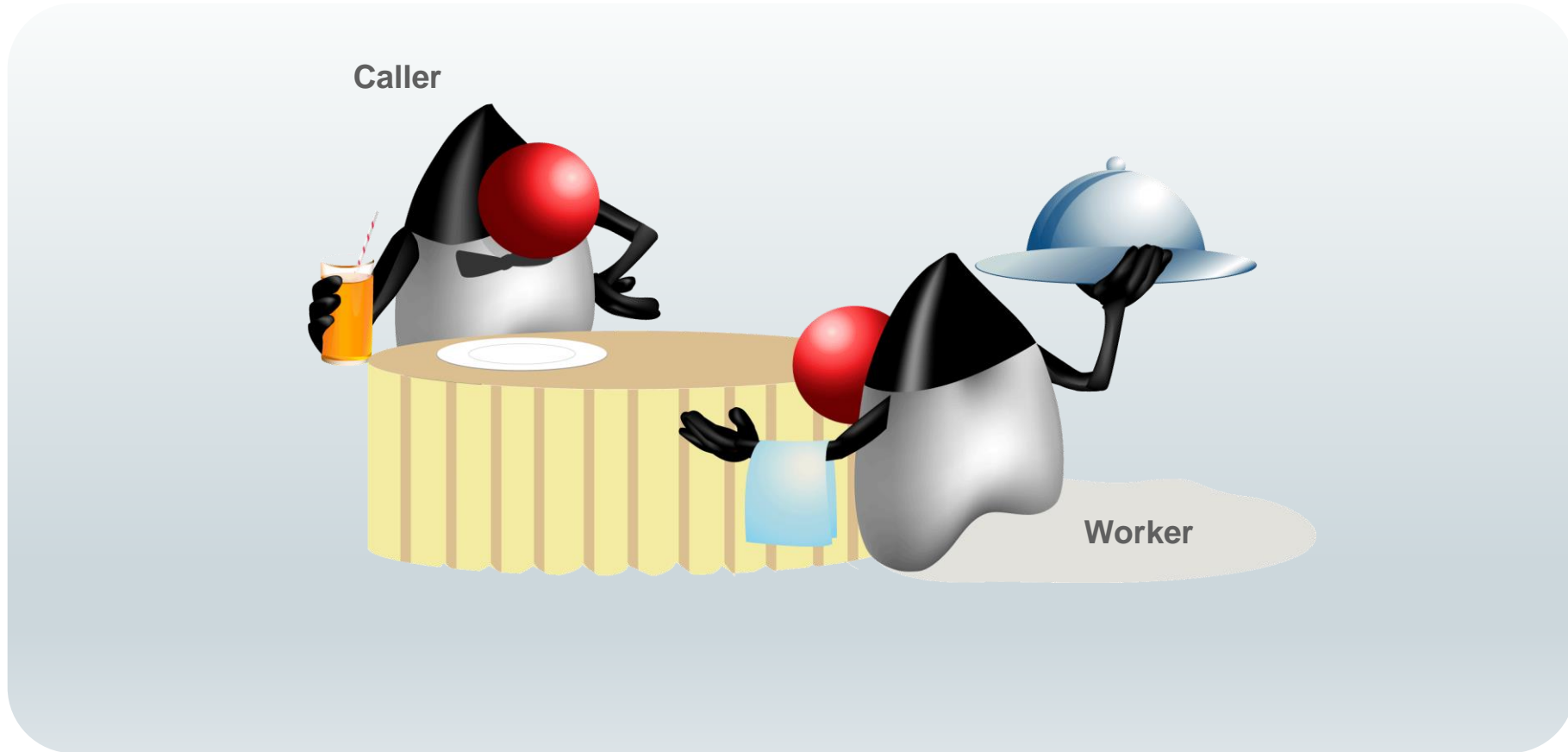
Dot operator

Method

Output:

```
Item description:-description required-  
Color Code: U  
Item price: 0.0
```

Caller and Worker Methods



A Constructor Method

A constructor method is a special method that is invoked when you create an object instance.

- It is called by using the `new` keyword.
- Its purpose is to instantiate an object of the class and store the reference in the reference variable.

```
Shirt myShirt = new Shirt();
```

Constructor method is called.

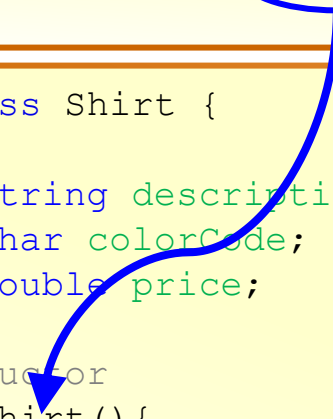
- It has a unique method signature.

```
<modifier> ClassName()
```

Writing and Calling a Constructor

```
1 public static void main(String[] args) {  
2     Shirt myShirt = new Shirt()  
3 }
```

```
1 public class Shirt {  
2     //Fields  
3     public String description;  
4     public char colorCode;  
5     public double price;  
6  
7     //Constructor  
8     public Shirt() {  
9         description = "--description required--";  
10        colorCode = 'U'  
11        price = 0.00;  
12    }  
13  
14    //Methods  
15    public void display() {  
16        System.out.println("Shirt description:" + description);  
17        System.out.println("Color Code: " + colorCode);  
18        System.out.println("Shirt price: " + price);  
19    }...
```



Calling a Method in the Same Class

```
1 public class Shirt {
2     public String description;
3     public char colorCode;
4     public double price;
5
6     public Shirt() {
7         description = "--description required--";
8         colorCode = 'U'
9         price = 0.00;
10
11         display();           //Called normally
12         this.display();      //Called using the 'this' keyword
13     }
14
15     public void display() {
16         System.out.println("Shirt description:" + description);
17         System.out.println("Color Code: " + colorCode);
18         System.out.println("Shirt price: " + price);
19     }
20 ...
```

Topics

- Using constructors and methods
- **Method arguments and return values**
- Using static methods and variables
- Understanding how arguments are passed to a method
- Overloading a method



Method Arguments and Parameters

- An **argument** is a value that is passed during a method call:

```
Calculator calc = new Calculator();  
double denominator = 2.0  
  
calc.calculate(3, denominator);           //should print 1.5
```

Arguments

- A **parameter** is a variable defined in the method declaration:

```
public void calculate(3int x, 2.0double y){  
    System.out.println(x/y);  
}
```

Parameters

Method Parameter Examples

- Methods may have any number or type of parameters:

```
public void calculate0(){  
    System.out.println("No parameters");  
}
```

```
public void calculate1(int x){  
    System.out.println(x/2.0);  
}
```

```
public void calculate2(int x, double y){  
    System.out.println(x/y);  
}
```

```
public void calculate3(int x, double y, int z){  
    System.out.println(x/y +z);  
}
```

Method Return Types

- Variables can have values of many different types:

`short` `int` `double` `long` `char` `float` `byte`
`String` `boolean` `int[]` `shirt`

- Method calls can also return values of many different types:

`short` `int` `double` `long` `char` `float` `byte`
`String` `boolean` `int[]` `shirt`

- How to make a method return a value:
 - Declare the method to be a non-void return type.
 - Use the keyword `return` within a method, followed by a value.

Method Return Types Examples

- Methods must **return** data that matches their return type:

```
public void printString() {  
    System.out.println("Hello");  
}
```

Void methods cannot
return values in Java.

```
public String returnString() {  
    return("Hello");  
}
```

```
public int sum(int x, int y) {  
    return(x + y);  
}
```

```
public boolean isGreater(int x, int y) {  
    return(x > y);  
}
```

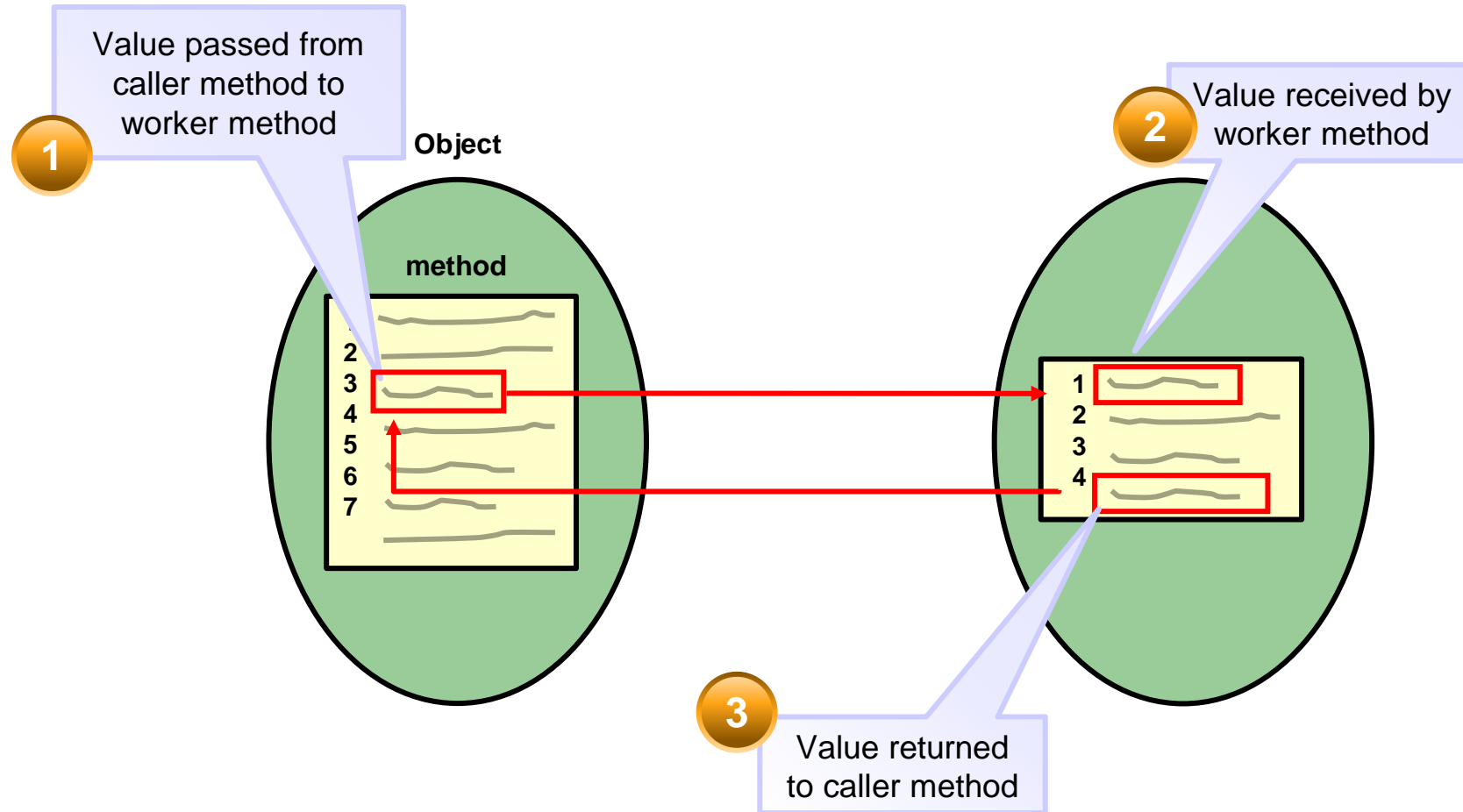
Method Return Animation

- The following code examples produce equivalent results:

```
public static void main(String[] args){  
    int num1 = 1, num2 = 2;  
    int result = num1 + num2;  
    System.out.println(result);  
}
```

```
public static void main(String[] args){  
    int num1 = 1, num2 = 2;  
    int result = sum(num1, num2);  
    System.out.println(result);  
}  
public int sum(int x, int y){  
    return(x + y);  
}
```

Passing Arguments and Returning Values



More Examples

```
1 public void setCustomerServices() {  
2     String message = "Would you like to hear about "  
3         + "special deals in your area?";  
4     if (cust.isNewCustomer()) {  
5  
6         cust.sendEmail(message);  
7     }  
8 }
```

```
1 public class Customer{  
2     public boolean isNew;  
3  
4     public boolean isNewCustomer() {  
5         return isNew;      ———— Return a boolean  
6     }  
7     public void sendEmail(String message) {  
8         // send email      ———— String argument required  
9     }  
10 }
```

Code Without Methods

```
1  public static void main(String[] args){
2      Shirt shirt01 = new Shirt();
3      Shirt shirt02 = new Shirt();
4      Shirt shirt03 = new Shirt();
5      Shirt shirt04 = new Shirt();
6
7      shirt01.description = "Sailor";
8      shirt01.colorCode = 'B';
9      shirt01.price = 30;
10
11     shirt02.description = "Sweatshirt";
12     shirt02.colorCode = 'G';
13     shirt02.price = 25;
14
15     shirt03.description = "Skull Tee";
16     shirt03.colorCode = 'B';
17     shirt03.price = 15;
18
19     shirt04.description = "Tropical";
20     shirt04.colorCode = 'R';
21     shirt04.price = 20;
22 }
```

Better Code with Methods

```
1  public static void main(String[] args){
2      Shirt shirt01 = new Shirt();
3      Shirt shirt02 = new Shirt();
4      Shirt shirt03 = new Shirt();
5      Shirt shirt04 = new Shirt();
6
7      shirt01.setFields("Sailor", 'B', 30);
8      shirt02.setFields("Sweatshirt", 'G', 25);
9      shirt03.setFields("Skull Tee", 'B', 15);
10     shirt04.setFields("Tropical", 'R', 20);
11 }
```

```
1  public class Shirt {
2      public String description;
3      public char colorCode;
4      public double price;
5
6      public void setFields(String desc, char color, double price){
7          this.description = desc;
8          this.colorCode = color;
9          this.price = price;
10     }
11     ...
```

Even Better Code with Methods

```
1 public static void main(String[] args){
2     Shirt shirt01 = new Shirt("Sailor", "Blue", 30);
3     Shirt shirt02 = new Shirt("SweatShirt", "Green", 25);
4     Shirt shirt03 = new Shirt("Skull Tee", "Blue", 15);
5     Shirt shirt04 = new Shirt("Tropical", "Red", 20);
6 }
```

```
1 public class Shirt {
2     public String description;
3     public char colorCode;
4     public double price;
5
6     //Constructor
7     public Shirt(String desc, String color, double price){
8         setFields(desc, price);
9         setColor(color);
10    }
11    public void setColor (String theColor){
12        if (theColor.length() > 0)
13            colorCode = theColor.charAt(0);
14    }
15 }
16 }
```

Variable Scope

```
1 public class Shirt {  
2     public String description;  
3     public char colorCode;  
4     public double price;  
5  
6     public void setColor (String theColor) {  
7         if (theColor.length() > 0)  
8             colorCode = theColor.charAt(0);  
9     }  
10 }  
11  
12 public String getColor() {  
13     return theColor; //Cannot find symbol  
14 }  
15  
16 }
```

Instance variable (field)

Local variable

Scope of theColor

Not scope of theColor

Advantages of Using Methods

Methods:

- Are reusable
- Make programs shorter and more readable
- Make development and maintenance quicker
- Allow separate objects to communicate and to distribute the work performed by the program

Exercise 8-1: Declare a `setColor` Method

1. Open the project `Exercise_08-1` in NetBeans.

In the `Item` class:

2. Declare a `setColor` method that takes a `char` as an argument (a color code) and returns a `boolean`. Return `false` if the `colorCode` is ' ' (a single space). Otherwise, assign the `colorCode` to the `color` field and return `true`.

In the `ShoppingCart` class:

3. Call the `setColor` method on `item1`. If it returns `true`, print `item1.color`. If it returns `false`, print an invalid color message.
4. Test the `setColor` method with both a valid color and an invalid one.



Topics

- Using constructors and methods
- Method arguments and return values
- **Using static methods and variables**
- Understanding how arguments are passed to a method
- Overloading a method



Static Methods and Variables

The `static` modifier is applied to a method or variable.

It means the method/variable:

- Belongs to the *class* and is shared by all objects of that class
- Is *not unique* to an object instance
- Can be accessed without instantiating the class

Comparison:

- A **static variable** is shared by all objects in a class.
- An **instance variable** is unique to an individual object.

Example: Setting the Size for a New Item

```
1 public class ItemSizes {  
2     static final String mSmall = "Men's Small";  
3     static final String mMed = "Men's Medium";  
4 }
```

Passing the static mMed variable
to the setSize method

```
Item item1 = new Item();  
item1.setSize(ItemSizes.mMed);
```

```
1 public class Item {  
2     public String size;  
3     public void setSize(String sizeArg) {  
4         this.size = sizeArg;  
5     }  
6 }
```

Creating and Accessing Static Members

- To create a static variable or method:

```
static String mSmall;  
static void setMSmall(String desc);
```

- To access a static variable or method:

- From another class

```
ItemSizes.mSmall;  
ItemSizes.setMSmall("Men's Small");
```

- From within the class

```
mSmall;  
setMSmall("Men's Small");
```

When to Use Static Methods or Fields

- Performing the operation on an individual object or associating the variable with a specific object type is not important.
- Accessing the variable or method before instantiating an object is important.
- The method or variable does not logically belong to an object, but possibly belongs to a utility class, such as the `Math` class, included in the Java API.
- Using constant values (such as `Math.PI`)

Some Rules About Static Fields and Methods

- Instance methods can access static methods or fields.
- Static methods cannot access instance methods or fields. Why?

```
1 public class Item{
2     int itemID;
3     public Item() {
4         setId();
5     }
6     static int getID() {
7         // whose itemID??
8     }
```

Static Fields and Methods Versus Instance Fields and Methods

```
public class Item{
    static int staticItemID;
    int instanceItemID;
    static main(){
        Item item01 = new Item();
```

- 1 staticItemID = 6; ✓
 - 2 instanceItemID = 3 ✗
 - 3 showItemID(); ✗
 - 4 item01.showItemID(); ✓
- ```
 }
 showItemID(){
 ...println(staticItemID);
 ...println(instanceItemID);
 }
}
```

Object (instance)  
referenced by item01.

```
static int/staticItemID;
int instanceItemID;
static main(){ ... }
```

```
showItemID(){
```

- 5 ...println(staticItemID); ✓
  - 6 ...println(instanceItemID); ✓
- ```
}
```

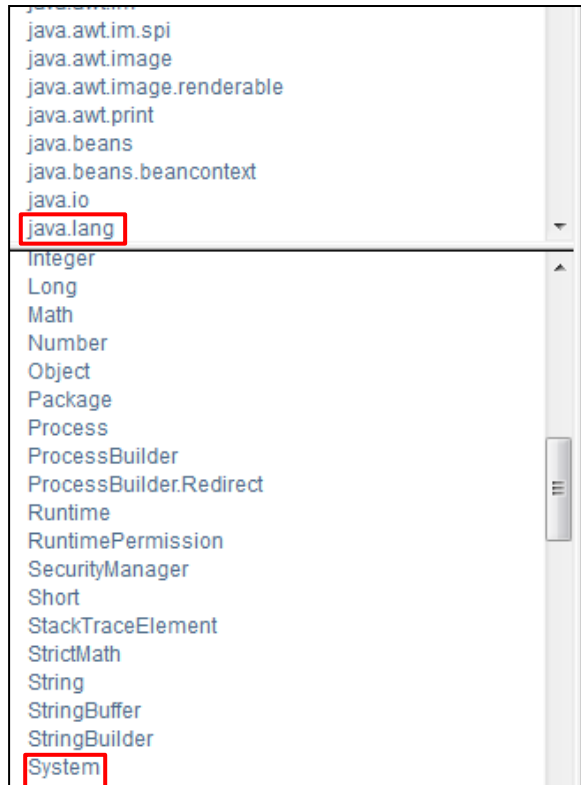
Other instances
of Item

Static Methods and Variables in the Java API

Examples:

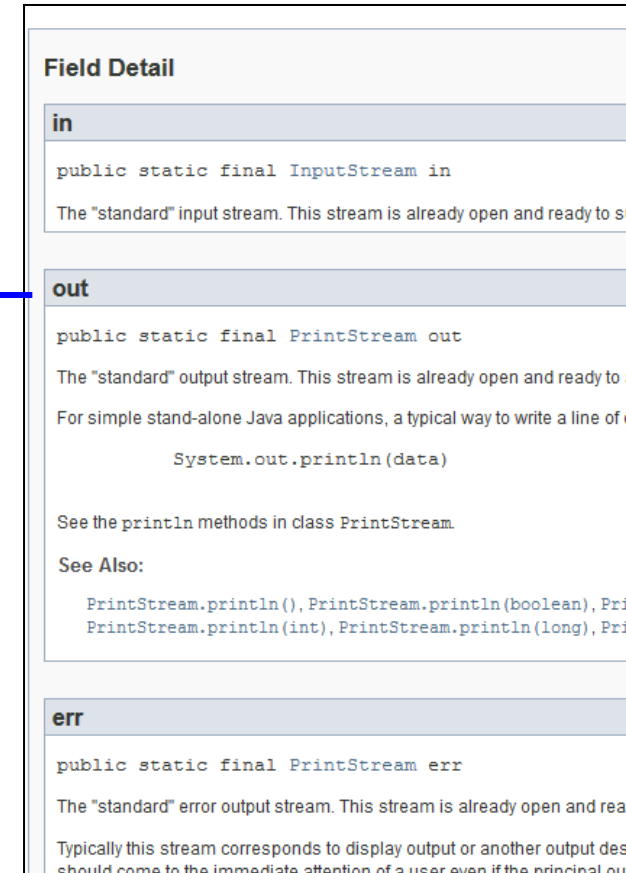
- Some functionality of the `Math` class:
 - Exponential
 - Logarithmic
 - Trigonometric
 - Random
 - Access to common mathematical constants, such as the value `PI` (`Math.PI`)
- Some functionality of the `System` class:
 - Retrieving environment variables
 - Access to the standard input and output streams
 - Exiting the current program (`System.exit` method)

Examining Static Variables in the JDK Libraries



System is a class in
`java.lang`.

out is a static
field of System
and contains
and is an
object
reference to a
PrintStream
object.



Using Static Variables and Methods: `System.out.println`

java.lang
Class System
java.lang.Object
java.lang.System

```
public final class System
extends Object
```

Field Summary

Fields

Modifier and Type	Field and Description
static <code>PrintStream</code> <code>err</code>	The "standard" error output stream.
static <code>InputStream</code> <code>in</code>	The "standard" input stream.
static <code>PrintStream</code> <code>out</code>	The "standard" output stream.

The field, `out`, on `System` is of type `PrintStream`.

Some of the methods of `PrintStream`

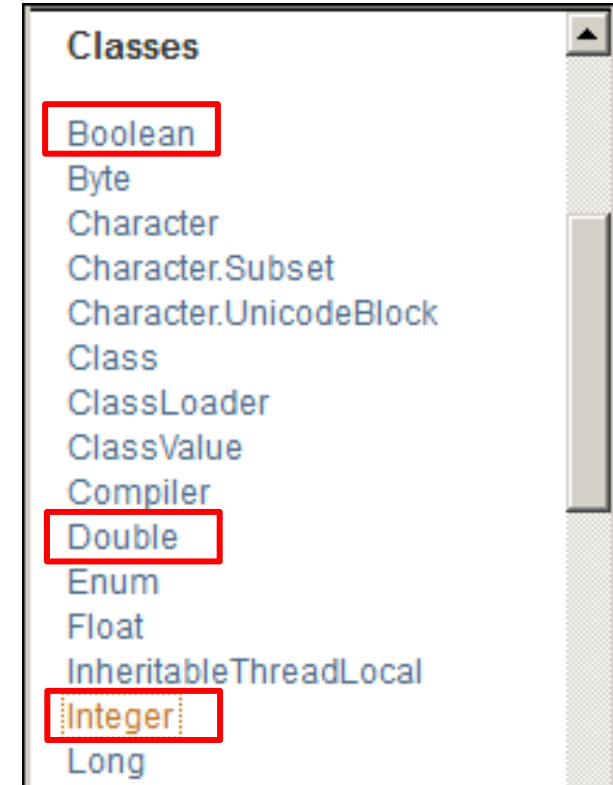
void	<code>print(Object obj)</code> Prints an object.
void	<code>print(String s)</code> Prints a string.
<code>PrintStream</code>	<code>printf(Locale l, String format, Object... args)</code> A convenience method to write a formatted string to this output
void	<code>println(double x)</code> Prints a double and then terminate the line.
void	<code>println(float x)</code> Prints a float and then terminate the line.
void	<code>println(int x)</code> Prints an integer and then terminate the line.
void	<code>println(long x)</code> Prints a long and then terminate the line.
void	<code>println(Object x)</code> Prints an Object and then terminate the line.
void	<code>println(String x)</code> Prints a String and then terminate the line.

More Static Fields and Methods in the Java API

Java provides wrapper classes for each of the primitive data types.

- **Boolean**: Contains a single field of type `boolean`
- **Double**: Contains a single field of type `double`
- **Integer**: Contains a single field of type `int`

They also provide utility methods to work with the data.



Converting Data Values

- Methods often need to convert an argument to a different type.
- Most of the object classes in the JDK provide various conversion methods.

Examples:

- Converting a String to an `int`

```
int myInt1 = Integer.parseInt(s_Num);
```

- Converting a String to a `double`

```
double myDb1 = Double.parseDouble(s_Num);
```

- Converting a String to `boolean`

```
boolean myBool = Boolean.valueOf(s_Bool);
```

Topics

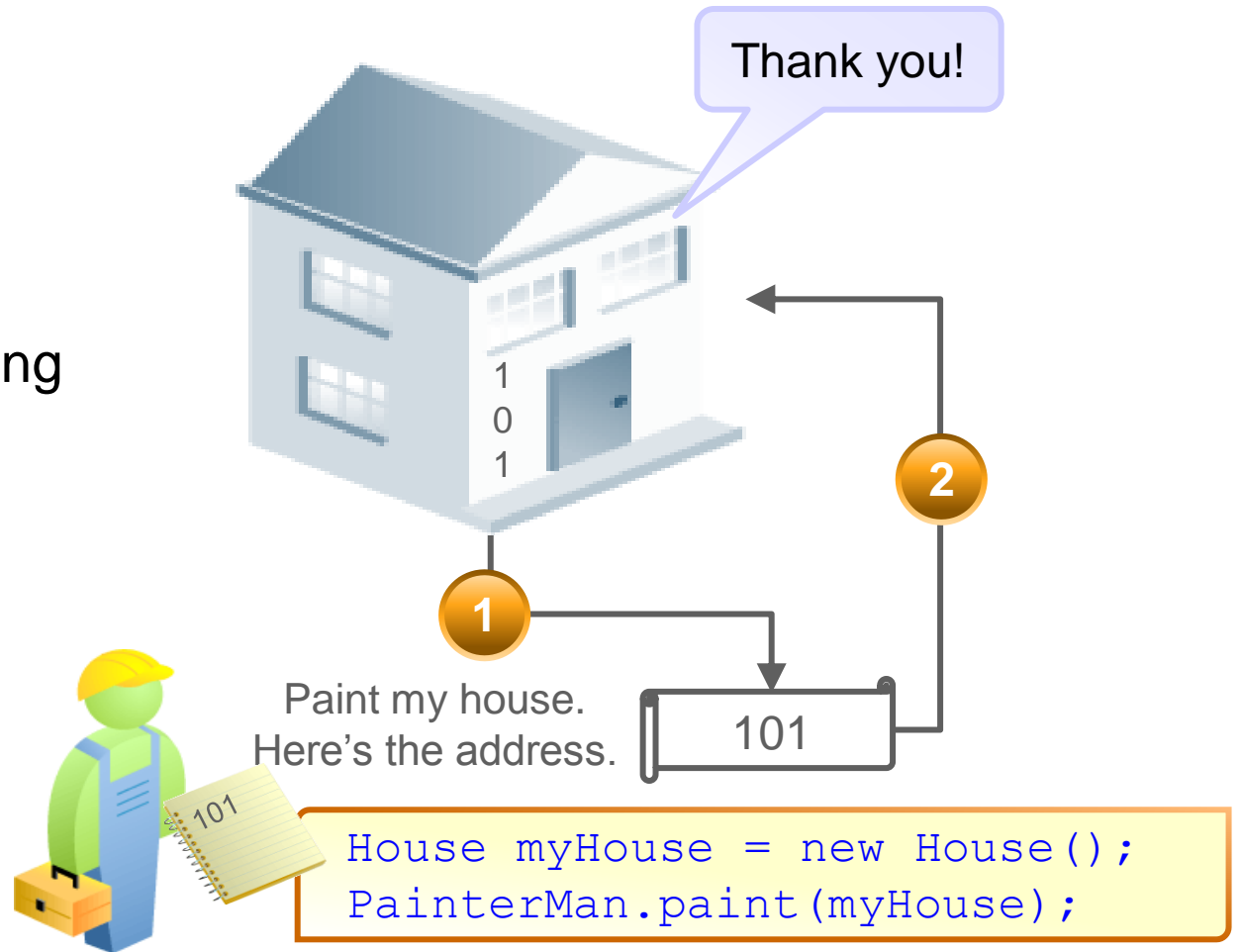
- Using constructors and methods
- Method arguments and return values
- Using static methods and variables
- Understanding how arguments are passed to a method
- Overloading a method



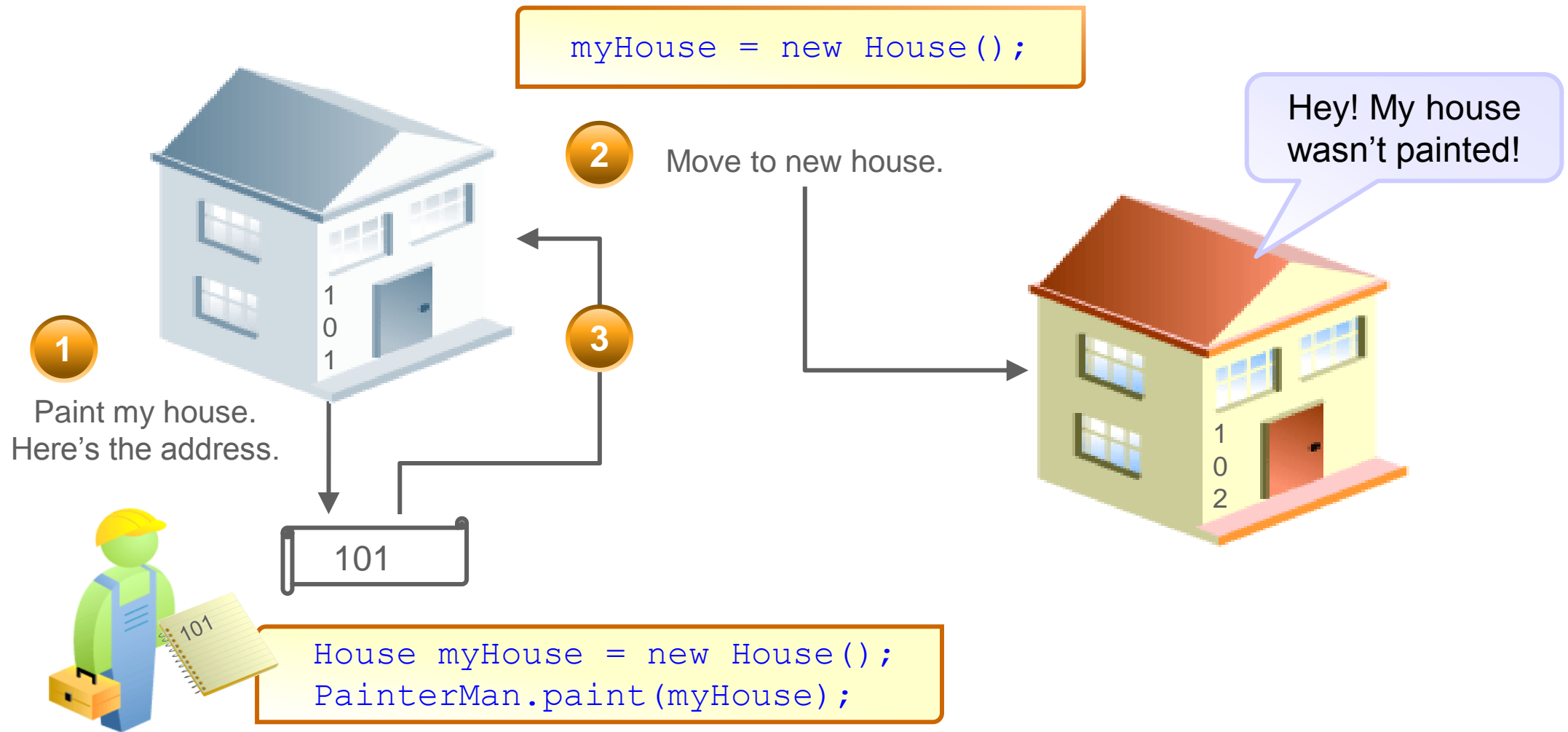
Passing an Object Reference

An object reference is similar to a house address. When it is passed to a method:

- The object itself is not passed
- The method can access the object using the reference
- The method can act upon the object



What If There Is a New Object?



A Shopping Cart Code Example

```
1 public class ShoppingCart {  
2     public static void main (String[] args) {  
3         Shirt myShirt = new Shirt();  
4         System.out.println("Shirt color: " + myShirt.colorCode);  
5         changeShirtColor(myShirt, 'B');  
6         System.out.println("Shirt color: " + myShirt.colorCode);  
7     }  
8     public static void changeShirtColor(Shirt theShirt, char color) {  
9         theShirt.colorCode = color;    }  
10 }
```

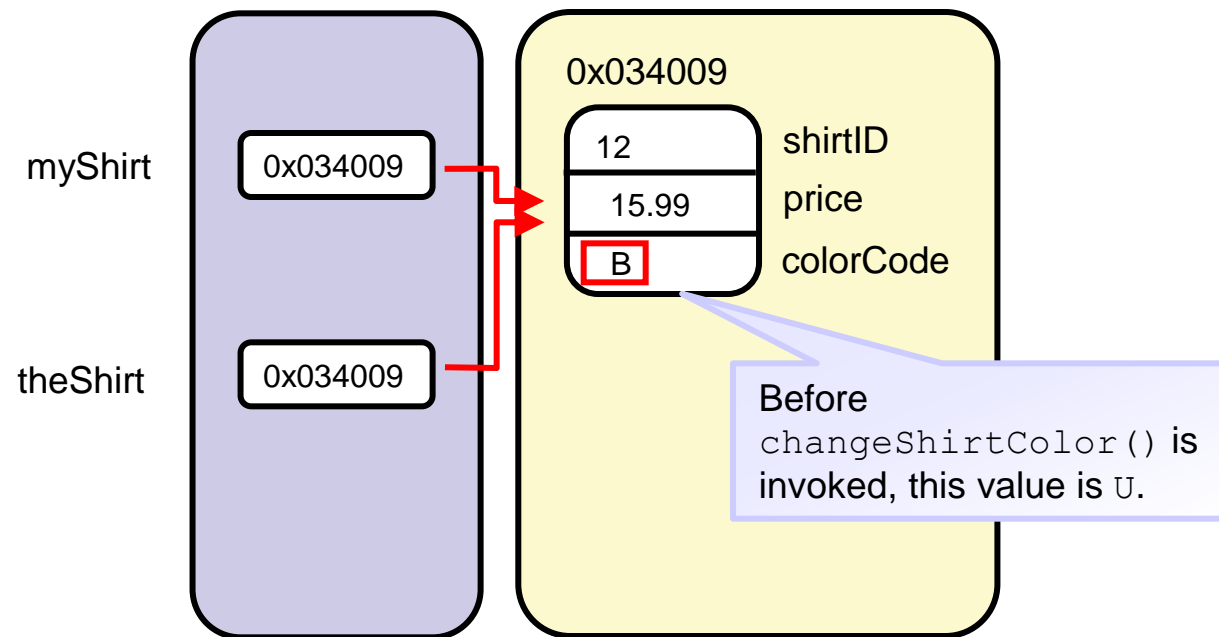
theShirt is a new reference of type Shirt.

Output:

```
Shirt color: U  
Shirt color: B
```

Passing by Value

```
Shirt myShirt = new Shirt();  
changeShirtColor(myShirt, 'B');
```



Reassigning the Reference

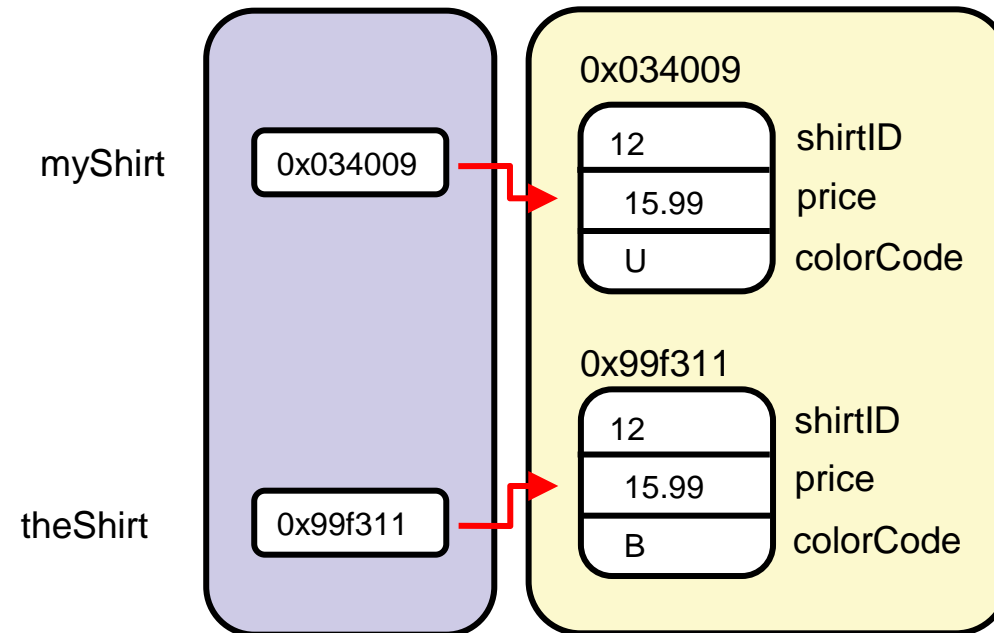
```
1 public class ShoppingCart {  
2     public static void main (String[] args) {  
3         Shirt myShirt = new Shirt();  
4         System.out.println("Shirt color: " + myShirt.colorCode);  
5         changeShirtColor(myShirt, 'B');  
6         System.out.println("Shirt color: " + myShirt.colorCode);  
7     }  
9     public static void changeShirtColor(Shirt theShirt, char color) {  
10         theShirt = new Shirt();  
11         theShirt.colorCode = color;  
12     }
```

Output:

```
Shirt color: U  
Shirt color: U
```

Passing by Value

```
Shirt myShirt = new Shirt();  
changeShirtColor(myShirt, 'B');
```



Topics

- Using constructors and methods
- Method arguments and return values
- Using static methods and variables
- Understanding how arguments are passed to a method
- Overloading a method



Method Overloading

Overloaded methods:

- Have the same name
- Have different signatures
 - The **number** of parameters
 - The **types** of parameters
 - The **order** of parameters
- May have different functionality or similar functionality
- Are widely used in the foundation classes

Using Method Overloading

```
1 public final class Calculator {
2
3     public static int sum(int num1, int num2) {
4         System.out.println("Method One");
5         return num1 + num2;
6     }
7
8     public static float sum(float num1, float num2) {
9         System.out.println("Method Two");
10        return num1 + num2;
11    }
12    public static float sum(int num1, float num2) {
13        System.out.println("Method Three");
14        return num1 + num2;
15    }
```

The method type

The method signature

Using Method Overloading

```
1  public class CalculatorTest {  
2  
3      public static void main(String[] args) {  
4  
5          int totalOne = Calculator.sum(2, 3);  
6          System.out.println("The total is " + totalOne);  
7  
8          float totalTwo = Calculator.sum(15.99F, 12.85F);  
9          System.out.println(totalTwo);  
10  
11         float totalThree = Calculator.sum(2, 12.85F);  
12         System.out.println(totalThree);  
13     }  
14 }
```

Method Overloading and the Java API

Method	Use
<code>void println()</code>	Terminates the current line by writing the line separator string
<code>void println(boolean x)</code>	Prints a boolean value and then terminates the line
<code>void println(char x)</code>	Prints a character and then terminates the line
<code>void println(char[] x)</code>	Prints an array of characters and then terminates the line

Exercise 8-2: Overload a `setItemFields` Method, Part 1

1. Open the project `Practice_08-2` in NetBeans.

In the `Item` class:

2. Write a `setItemFields` method that takes three arguments and assigns them to the `desc`, `quantity`, and `price` fields. The method returns `void`.
3. Create an overloaded `setItemFields` method to take four arguments and return an `int`. The method assigns all four fields. A ' ' (a single space) is an invalid value for a `colorCode` argument.
 - If the `colorCode` argument is invalid, return -1 without assigning the value.
 - If the `colorCode` is valid, assign the `colorCode` field and then assign the remaining fields by calling the three-argument method.



Exercise 8-2: Overload a `setItemFields` Method, Part 2

In the `ShoppingCart` class:

4. Call the 3-argument `setItemFields` method and then call `item1.displayItem()`.
5. Call the 4-argument `setItemFields` method. Check the return value.
 - If the return value < 0 , print an invalid color code message.
 - Otherwise, call `displayItem()`.



Quiz



Which method corresponds to the following method call?

```
myPerson.printValues(100, 147.7F, "lavender");
```

- a. `public void printValues (int i, float f)`
- b. `public void printValues (i, float f, s)`
- c. `public void printValues (int i, float f, String s)`
- d. `public void printValues (float f, String s, int i)`



Summary

In this lesson, you should have learned how to:

- Add an argument to a method
- Instantiate a class and call a method
- Overload a method
- Work with static methods and variables
- Convert data values using `Integer`, `Double`, and `Boolean` object types



Practices Overview

- 8-1: Using Methods
- 8-2: Creating Game Data Randomly
- 8-3: Creating Overloaded Methods

