Developing and Using Methods

Objectives

- Describe the advantages of methods and define worker and calling methods
- Declare and invoke a method
- Compare object and static methods
- Use overloaded methods

Creating methods

- Syntax of method declaration

```
[modifiers] return_type method_identifier
  ([arguments]) { method_code_block }
```

- Modifiers: keywords that modify the way methods are used (optional)
- return_type: only one item (type) or void
- method_identifier: name of the method
- ([arguments]): optional list of variables
- method_code_block: statements

Creating methods (2)

- Basic form of a method accepts no arguments and returns nothing
- example

```
public void displayShirtInformation()
{
       System.out.println("Shirt ID:"+description);
          ...
          // end of display method
}
```

Invoking a method

- To invoke a method in a different class, you can use the dot operator (.) with an object reference variable.

```
public class ShirtTest
{
    public static void main(String args[])
    {
        Shirt myShirt;
        myShirt = new Shirt();
        myShirt.displayShirtInformation();
    }
}
```

- The `main` method is the *calling* method, the `display` method is the *worker* method.
- When a calling method calls a worker method, the calling method stops execution until the worker method is done

Invoking a method in same class

- Just include the name of the worker method and its arguments

```
public class Elevator
{
    public boolean doorOpen = false;
    public int currentFloor = 1;

    public final int TOP_FLOOR = 5;
    public final int BOTTOM_FLOOR = 1;

    public void openDoor() {
        System.out.println("Opening door.");
        doorOpen = true;
        System.out.println("Door is open");
    }

    public void closeDoor(){
        System.out.println("Closing door.");
        doorOpen = false;
        System.out.println("Door is closed");
    }
```

Invoking a method in same class (2)

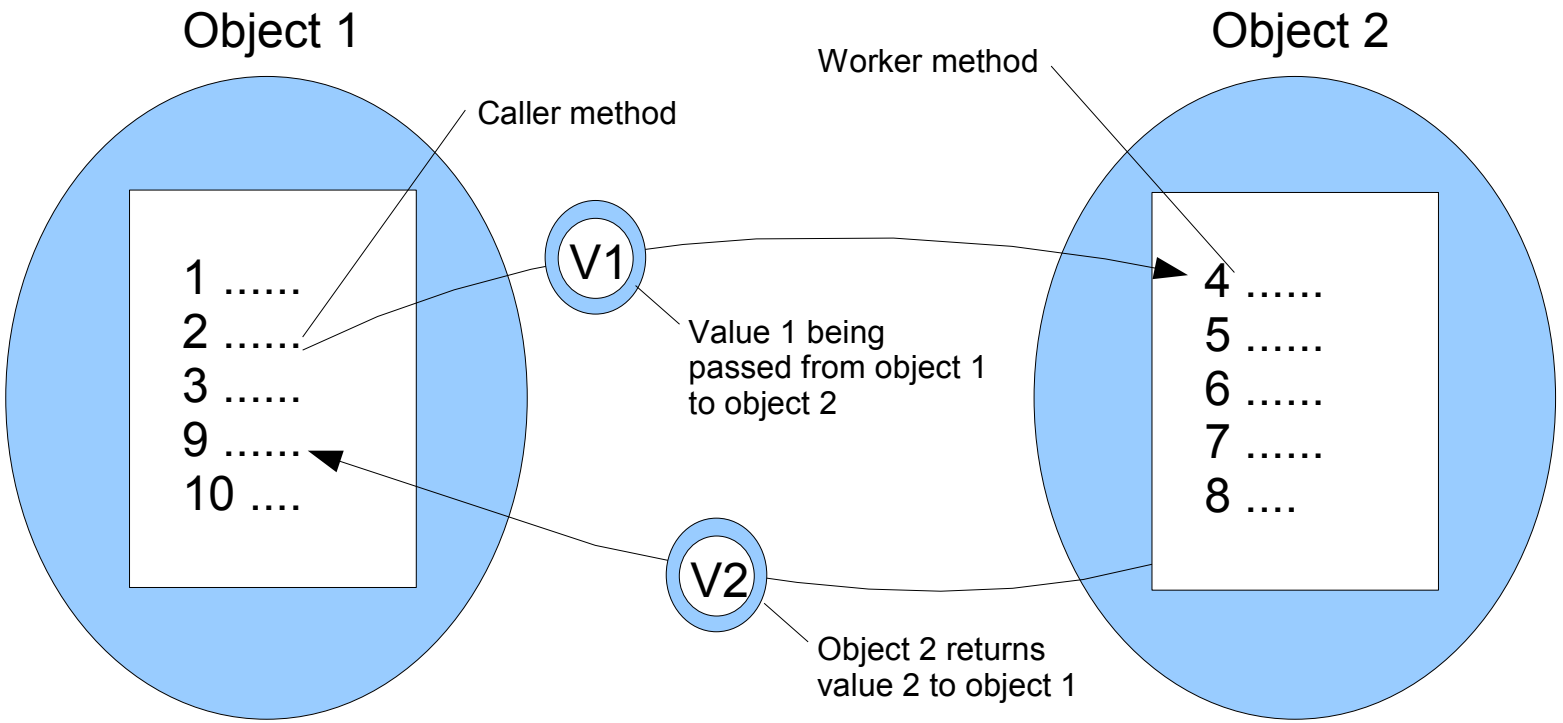
```
    public void goUp() {
        System.out.println("Going up one floor.");
        currentFloor++;
        System.out.println("Floor: "+currentFloor);
    }

    public void goDown() {
        System.out.println("Going down one floor.");
        currentFloor--;
        System.out.println("Floor: "+currentFloor);
    }

    public void setFloor(int desiredFloor) {
        while (currentFloor != desiredFloor) {
            if (currentFloor < desiredFloor) {
                goUp();
            }
            else {
                goDown();
            }
    }

    public int getFloor() { return currentFloor; }

    public boolean checkDoorStatus { return doorOpen; }
}
```

Guidelines for invoking methods

- There is no limit to the number of method calls that a calling method can make
- The calling method and the worker method can be in the same or in different classes
- The way you invoke the worker method is different, depending on whether it is in the same class or a different class from the calling method
- You can invoke methods in any order. Methods do not need to be completed in the order in which they are listed in the class where they are declared

Passing arguments and returning values

- Methods can be invoked by a calling method with a list of arguments
- Methods can return a value to the calling method that can be used in the calling method



Declaring methods with Arguments

- Example

```
public void setFloor(int desiredFloor) {
    while (currentFloor != desiredFloor) {
        if (currentFloor < desiredFloor) {
            goUp();
        }
        else {
            goDown();
        }
    }
}

public void multiply(int numberOne, int numberTwo);
```

- The main method
  - Arguments are values passed from the command line

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

java ShirtTest 12.99 R
```

Invoking methods with arguments

- To pass arguments from one method to another, include the arguments in the parentheses of the method call
  - You can pass literal or variables as arguments

```
public class ElevatorTest {
    public static void main(String args[]) {
        Elevator myElevator = new Elevator();

        myElevator.openDoor();
        myElevator.closeDoor();
        myElevator.goUp();
        myElevator.goUp();
        myElevator.goUp();
        myElevator.openDoor();
        myElevator.closeDoor();
        myElevator.goDown();
        myElevator.openDoor();
        myElevator.closeDoor();
        myElevator.goDown();

        myElevator.setFloor(myElevator.TOP_FLOOR);

        myElevator.openDoor();
    }
}
```

Declaring methods with return values

- `void` is used for methods that do not return a value
- To declare a method that returns a value, place the type of value in front of the method identifier

```
public int sum(int numberOne, int numberTwo)
```

  - A method can return only one value, but can accept multiple arguments
- To return a value from a method, use the `return` keyword

```
public int sum(int numberOne, int numberTwo) {
    int return = numberOne + numberTwo;

    return result;
}
```

Receiving return values

- If you invoke a method that returns a value you can use the return value in the calling method

```
public class ElevatorTestTwo {
    public static void main(String args[]) {
        Elevator myElevator = new Elevator();

        myElevator.openDoor();
        myElevator.closeDoor();
        myElevator.goUp();
        myElevator.goUp();
        myElevator.goUp();
        myElevator.openDoor();
        myElevator.closeDoor();
        myElevator.goDown();
        myElevator.openDoor();
        myElevator.closeDoor();
        myElevator.goDown();

        int curFloor = myElevator.getFloor();

        System.out.println("CurrentFloor: " + curFloor);

        myElevator.setFloor(curFloor + 1);

        myElevator.openDoor();
    }
}
```

Advantages of method use

- Methods make programs more readable and easier to maintain
- Methods make development and maintenance quicker
- Methods are central to reusable software
- Methods allow separate objects to communicate and to distribute the work performed by the program

Creating `static` methods and variables

- Methods and variables that are unique to an instance are called *instance methods* and *instance variables*.

- Methods such as the main method do not require object instantiation. They are called *class methods* or *static methods*. You can invoke them without creating an object first.

Declaring and invoking static methods

- Static methods are declared using the `static` keyword.

```
static Properties getProperties()
```

- Because static or class methods are not part of every object instance, **you should not** use an object reference variable.
- To invoke a static method, the class name is used. The syntax is the following
  - *Classname.method();*

Declaring and invoking static methods

- The following method could be added to the Shirt class to convert numerical sizes to sizes
  - It does not directly use any attributes of the Shirt class

```
public static char convertShirtSize(int numericalSize) {
    if (numericalSize < 10) {
        return 'S';
    }
    else if (numericalSize < 14) {
        return 'M';
    }
    else if (numericalSize < 18) {
        return 'L';
    }
    else {
        return 'X';
    }
}
```

- Invoking the method

```
char size = Shirt.convertShirtSize(16);
```

Declaring and accessing `static` variables

- You can also use the static keyword to declare that there can only be one copy of the variable in memory associated with a class, not a copy for each object.
  - `static double salesTax = 8.25;`
- As with static methods, you should use the class name to access a static variable. The syntax is
  - *Classname.variable;*
  - `Double myPI = Math.PI;`
- You can have both the `static` and `final` modifiers to indicate that there is only one copy of the variable and that its contents cannot be changed
  - `static final int PI`

Static methods and variables in the Java API

- Certain Java class libraries such as the `System` class contain only static methods and variables.
  - The System class contains utility methods for handling operating system specific tasks.
  - For example, the `System.getProperties()` method gets information about the computer you are using
- There are several classes in the API that are utility classes. They contain `static` methods that are useful for objects of all types.
  - The `Math` class
  - The `System` class

When to declare a `static` method or variable

- You might consider declaring a method or variable `static` if
  - Performing an operation on an individual object or associating the variable with a specific object type is not important
  - Accessing the variable or method before instantiating and object is important
  - The method or variable does not logically belong to an object, but possibly belongs to a utility class.

Using method overloading

- There can be several methods in a class that have the same name but different arguments. This concept is called *method overloading*.
  - Methods with the same name can be distinguished by their arguments
- For example, you may want to create a class that contains several methods with the same name to add two numbers of different types

Using method overloading (2)

```
public class Calculator {
    public int sum(int numberOne, int numberTwo) {
        System.out.println("Method One");
        return numberOne + numberTwo;
    }

    public float sum(float numberOne, float numberTwo) {
        System.out.println("Method Two");
        return numberOne + numberTwo;
    }

    public float sum(int numberOne, float numberTwo) {
        System.out.println("Method Three");
        return numberOne + numberTwo;
    }
}
```

Using method overloading (3)

```
public class CalculatorTest {
    public static void main(String args[]) {
        Calculator myCalculator = new Calculator();

        int totalOne = myCalculator.sum(2,3);
        System.out.println(totalOne);

        int totalTwo = myCalculator.sum(15.99F,12.85F);
        System.out.println(totalTwo);

        int totalThree = myCalculator.sum(2,12.85F);
        System.out.println(totalThree);

    }
}
```

Method overloading and the Java API

- Many methods in the Java API are overloaded, including the `System.out.println` method
- Variations of the `println` method
  - `void println()`
  - `void println(boolean x)`
  - `void println(char x)`
  - `void println(char[] x)`
  - `void println(double x)`
  - `void println(float x)`
  - `void println(int x)`
  - `void println(long x)`
  - `void println(Object x)`
  - `void println(String x)`

Uses for method overloading

- When you write code, keep in mind that you must define overloaded methods if the actions the method completes must be performed on several types of data.
- You can also use overloading to create several methods with the same name but with a different number of parameters.
  - `public int sum(int numberOne, int numberTwo)`
  - `public int sum(int numberOne, int numberTwo, int numberThree)`
  - `public int sum(int numberOne, int numberTwo, int numberThree, int numberFour)`

Invoking overloaded methods

- The following example creates three object instances of the `ShirtTwo` class and uses their overloaded methods

```
public class ShirtTwoTest {
    public static void main(String args[]) {
        ShirtTwo shirtOne = new ShirtTwo();
        ShirtTwo shirtTwo = new ShirtTwo();
        ShirtTwo shirtThree = new ShirtTwo();

        shirtOne.setShirtInfo(100,”Button Down”, 12.99);
        shirtTwo.setShirtInfo(101,”Long Sleeve Oxford”, 27.99,
'G');
        shirtThree.setShirtInfo(102,”Short Sleeve T-Shirt”,
9.99, 'B');

        shirtOne.display();
        shirtTwo.display();
        shirtThree.display();
    }
}
```