Tutorial 3

Topics

- Methods
  - Define a method
  - Call a method
  - Call by value
  - Constructors
  - Static method
- Arrays
- Vectors
- Tutorial problems
- Design problems

Methods

In a class, each method performs a relatively autonomous task. This allows programmers to break down a complex problem into several small and manageable pieces. Once a method is written and successfully tested the programmer need not worry about its internal workings. Users simply use the method name and appropriate arguments to invoke the method whenever its functionality is required.

Each method has a

- Access identifier. Like variables, methods can be public or private (there are other access identifiers we will cover later in the term.) Private methods can only be accessed within the class. Public methods, on the other hand, can be accessed by anyone, both inside and outside the class.
- Return type. A method may return a value to the user. It could be a simple data type like int or another class. A void return type indicates that no value will be returned.
- Name. A method name must start with a letter, is case sensitive, and cannot be a Java reserved words. A good practice is to use descriptive names, e.g. setHeight
- Argument list. Arguments are input fields for a method. A parameter can be simple data type or another class

In Java, DIFFERENT methods in the SAME class can have the same name and return the same data type, but have a DIFFERENT set of parameters (this set of arguments is called the method's signature). This is called the "method overloading." The correct method will be called based on the type of the argument(s) and number of argument(s).

Although this paragraph is rather short, the concept of method overloading is very important and very useful in many Java programs. You shall see various method overloading examples in the next lectures and problem sets.
Define a Method

A method has to be defined before it can be invoked. Method definition provides the details of how to perform the tasks. Since each method is an autonomous chunk of code, it can define its own variables and methods.

Using our Box class, we will define 7 methods:

```java
public class Box {
    private double width;
    private double height;
    private double length;

    // Method to calculate the volume of a box
    public double volume() {
        return width*height*length;
    }

    // Method to set the value of the box's width
    public void setWidth(double w) {
        width = w;
    }

    // Method to set the value of the box's height
    public void setHeight(double h) {
        height = h;
    }

    // Method to set the value of the box's length
    public void setLength(double l) {
        length = l;
    }

    // Method to get the value of the box's width
    public double getWidth() {
        return width;
    }

    // Method to get the value of the box's height
    public double getHeight() {
        return height;
    }

    // Method to get the value of the box's length
    public double getLength() {
        return length;
    }
}
```
Using the methods of "Box" class:

To call the methods that we defined in the class Box, we use a . (dot operator). For instance:

```java
public class Example{
    public static void main (String args[]) {
        Box myFirstBox = new Box();       // creating the box
        myFirstBox.setWidth(7.5);
        myFirstBox.setHeight(6.97);
        myFirstBox.setLength(2);
        System.out.println("The volume is " + myFirstBox.volume());
    }
}
```

Output: The volume is 104.55

**Call by Value**

When a method is invoked, Java creates a temporary variable for each of its parameters and copies the value of that parameter to the temporary variable. In other words, the method gets just the value of that parameter, not the parameter itself. Any changes made inside the method will only affect the temporary variable, not the parameter passed to the method. This is called "call_by_value" in computer science terms.

Note that Java always uses call_by_value for parameter passing. This is more subtle than it appears at first glance, though, especially when objects are being passed, as explained below.

In the following example, we defined a method called increment which takes two arguments: an integer and a Box. Inside the method, we changed both arguments:

```java
void doubleWidth (double w, Box b) {
    w = w * 2;
    b.setWidth (w);
    System.out.println("Inside method: w = " + w + " width = " + b.getWidth());
}
```

Here is the main program:

```java
public static void main (String args[]) {
    Box myFirstBox = new Box();
    myFirstBox.setWidth(7.5);
    double w = myFirstBox.getWidth();
    ```
System.out.println("Before calling the method: w = "+w+" width = "+myFirstBox.getWidth());
Example ex = new Example();
ex.doubleWidth(w, myFirstBox);
System.out.println("After calling the method: w = "+w+" width = "+myFirstBox.getWidth());
}

Here is the output:

Before calling the method: w = 7.5 width = 7.5
Inside the method: w = 15.0 width = 15.0
Before calling the method: w = 7.5 width = 15.0

As we expected, the value of w didn’t change before and after calling the method.
Wait a minute: why did the width of myFirstBox change to 15 after calling the method?
Didn’t you just say that the method would not change the content of the parameters?

Well, let’s look at how the method was called.

ex.doubleWidth(w, myFirstBox);

So what is myFirstBox? Is it an object or a reference to the object (if you don’t know
the answer, check the previous tutorial.) You are right; it is a reference to the object
created by the new operator.

Inside the method, Java created a temporary variable b(which is also a reference) and
copied the content of myFirstBox to b. Now, both b and myFirstBox are references to
the same object. So when we changed the width of the object pointed by b, we also
changed the width of the object pointed by myFirstBox because they are the same
object.

For those who found it rather confusing, here is the bottom line.

- If the parameter is a primitive data type (int, double, etc.), changes made
  inside the method WILL NOT be carried back to the caller
- If the parameter is an object reference, changes made inside the method WILL
  be carried back to the caller

See the text, pp. 148-151 for a detailed example and further explanation.
A Special Method - Constructor

Constructors are methods that are used to CREATE an object of the class. They are automatically called when the new operator is used. A constructor doesn't have a return type; it has the same name as its class and can have any number of parameters.

One class may have many constructors. They are mostly used to initialize member variables of an object. For example, here is an example of the constructors for the Box class:

//Default constructor
public Box() {
    width = 0;
    height = 0;
    length = 0;
}

//Constructor with arguments
public Box (double w, double h, double l) {
    width = w;
    height = h;
    length = l;
}

Constructors are called in the following way:

Box b1 = new Box();
Box b2 = new Box(2.0, 2.5, 3.0);

Note that this is a classic case of method overloading. Both constructors have the same name (Box) but different set of arguments. As a result, they are different methods: object b1 has zero width, height, and length while b2 has a width of 2.0, height of 2.5, and length of 3.0.

If no constructor is defined for a class, the compiler creates a default constructor that takes no arguments. This default constructor will implicitly initialize the instance variables to their default values (0 for primitive numeric types, false for boolean and null for references).

Static method

When a method definition is preceded by the keyword static, it is called a static method. The main difference between regular class methods and static methods is that static methods operate on the class while regular methods operate on instances.
of the class. Therefore, static methods are called using their class name not the instance name.

For example:

```java
public static void printBox (Box b) {
    System.out.println(b.getWidth());
    System.out.println(b.getHeight());
    System.out.println(b.getLength());
}
```

The method printBox can now be called from a program as follows

```java
Box myBox = new Box (1.0, 2.0, 3.0);
Box.printBox(myBox);
```

Compare this with a call to the regular method volume():

```java
double result = myBox.volume();
```

### Arrays

An array is a set of values that have the same type. To refer to a particular element in the array, we specify the name of the array followed by the index number of that particular element in square brackets[]. Note that the first element of the array is indexed as 0, not 1.

In the following example, we want to create an array `c` which contains 6 integers (c[0] to c[5]).

```
c[0] 2
c[1] 5
```

```
c[2] 1
```

```
c[3] 43
```

```
c[4] 8
```

```
c[5] 66
```

Here is its declaration:

```java
int[] c = new int[6]; //Allocate memory for 6 integers in array c
c[0] = 2;
c[1] = 5;
......................
```

Or you can initialize and declare the array at the same time.

```java
int[] c = {2, 5, 1, 43, 8, 66};
```

Array can store not only primitive data types, but also objects. For example:
Box[] boxArray = new Box[3];

Every array object has an integer data member called length that returns the number of elements of that array. For example:

System.out.println(c.length); //outputs 6

**Vectors**

Arrays have two main disadvantages:

- The length of the array is fixed. It cannot increase or decrease as needed. The memory is allocated based on the length, even if the array does not use all the space.
- They can hold elements of only one type

To get around these limitations, Java created a class called Vector in the package java.util

Like an array, a vector is a collection of objects, but:

- A vector can increase its length automatically when needed.
- A vector can hold elements of different types as long as each is an object (but NOT a primitive data type; you can not put integers or doubles in a vector).
- A vector is accessed, added to, changed, and queried using methods of the Vector class

Here are some most commonly used methods:

- Vector(int initialCapacity)
- add(Object element)
- elementAt(int index)
- get(int index)
- isEmpty()
- remove(int index)
- size()

Note that the length of a vector v is v.size(), not v.length

Here is an example of using the Vector class

```java
import java.util.*;
class TestVector{
    public static void main (String[] args) {
        Vector bv = new Vector(2); //create a vector with the capacity of 2
        System.println("Capacity = " + bv.capacity() + " size = " + bv.size());
    }
}
```
```
Box b1 = new Box(1.0, 2.0, 3.0);
Box b2 = new Box();
Box b3 = new Box(0.5, 1.5, 2.5);
bv.add(b1); //add b1 to the end of vector
bv.add(b2); //add b2 to the end of vector
bv.add(b3); //add b3 to the end of vector
System.println("Capacity = " + bv.capacity() + " size = " + bv.size());
System.println("Width of the 1st element = " + ((Box)bv.firstElement()).getWidth());
System.println("Width of the 2nd element = " + ((Box)bv.elementAt(1)).getWidth());
System.println("Width of the last element = " + ((Box)bv.lastElement()).getWidth());
bv.remove(2);
System.println("Capacity = " + bv.capacity() + " size = " + bv.size());
```

Here is the output

```
Capacity = 2 size = 0
Capacity = 4 size = 3
Width of the 1st element = 1.0
Width of the 2nd element = 0.0
Width of the last element = 0.5
Capacity = 4 size 2
```

**Tutorial Problems**

**Constructors and Method Overloading**

1. Please write down the output of the following program.

```java
public class Item {
    public int value;
    public final int defaultValue = 5;
    public Item(int a, int b) {
        value = a * b;
    }
    public Item() {
        value = defaultValue;
    }

    public static void main(String args[]){
        Item a1 = new Item();
        Item a2 = new Item(2,2);
        Item a3 = a1;
```
System.out.println("Item1's value is "+a1.value);
System.out.println("Item2's value is "+a2.value);
System.out.println("Item3's value is "+a3.value);
}
}

Write the output of the above program in the space below:

Line 1:
Line 2:
Line 3:

Static Field

2. Modify the above class definition and add a data field called numOfItems, which keeps a record of how many items have been instantiated. Then print its value at the beginning and end of the main program. What is the value of numOfItems in the above example?

Array

3. Answer the following questions regarding an array called fractions.

a. Define a constant variable ARRAY_SIZE initialized to 5.
b. Declare an array with ARRAY_SIZE elements of type float and initialize all elements to 0.
c. Name the fourth element of the array.
d. Assign 1.5 to the fourth element of the array.
e. Sum all the elements of the array using a for repetition structure.

Design Problem

Write a Java program that helps MIT schedule its classrooms. Each classroom has a room number (e.g. 5-008), capacity and is available for use from 9am to 5pm in one hour blocks. Each course has a name, course ID, and a number of students. Define a set of Java classes, including their member data and methods, which represents this situation and provides the capabilities for someone to schedule the classrooms.

To create a schedule for a particular classroom, one needs the course ID, capacity, and time. If the time segment has been scheduled for another class, the program should print an error message. If room doesn’t have the necessary capacity, the program should print another error message.

Hint: think of the time segments as an array.