Advanced Programming

Class Parameters
Class Variables
Class Parameters

• All parameters in Java are *call-by-value* parameters
  – A parameter is a *local variable* that is set equal to the value of its argument
  – Therefore, any change to the value of the parameter cannot change the value of its argument

• Class type parameters appear to behave differently from primitive type parameters
  – They appear to behave in a way similar to parameters in languages that have the *call-by-reference* parameter passing mechanism
Class Parameters

• The value plugged into a class type parameter is a reference (memory address)
  – Therefore, the parameter becomes another name for the argument
  – Any change made to the object named by the parameter (i.e., changes made to the values of its instance variables) will be made to the object named by the argument, because they are the same object
  – Note that, because it still is a call-by-value parameter, any change made to the class type parameter itself (i.e., its address) will not change its argument (the reference or memory address)
• Recall that the following makes variable1 and variable2 two names for the same object:

    ToyClass variable1 = new ToyClass("Joe", 42);
    ToyClass variable2;
    variable2 = variable1;

• The parameter is a local variable that is set equal to the value of its argument. But if its argument is a variable of a class type, this copies a reference into the parameter. So, the parameter becomes another name for the argument, and any change made to the object named by the parameter is made to the object named by the argument, because they are the same object.
public class ToyClass {
    private String name;
    private int number;

    public ToyClass(String initialName, int initialNumberOf) {
        name = initialName;
        number = initialNumberOf;
    }

    public ToyClass() {
        name = "No name yet.";
        number = 0;
    }

    public void set(String newName, int newNumberOf) {
        name = newName;
        number = newNumberOf;
    }
}
public String toString()
{
    return (name + " " + number);
}

public static void changer(ToyClass aParameter)
{
    aParameter.name = "Hot Shot";
    aParameter.number = 42;
}

public boolean equals(ToyClass otherObject)
{
    return ( (name.equals(otherObject.name))
            && (number == otherObject.number) );
}
Parameters of a Class Type

Display 5.14  Parameters of a Class Type

```java
public class ClassParameterDemo {
    public static void main(String[] args) {
        ToyClass anObject = new ToyClass("Mr. Cellophane", 0);
        System.out.println(anObject);
        System.out.println("Now we call changer with anObject as argument.");
        ToyClass.changer(anObject);
        System.out.println(anObject);
    }
}
```

ToyClass is defined in Display 5.11.

Notice that the method changer changed the instance variables in the object anObject.

**Sample Dialogue**

Mr. Cellophane 0
Now we call changer with anObject as argument.
Hot Shot 42
Before anything:

We do not know what memory address (reference) is stored in the variable `anObject`. Let's say it is 3078. The exact number does not matter.

Someplace else in memory:

(continued)
anObject is plugged in for aParameter.
anObject and aParameter become two names for the same object.

Someplace else in memory:
ToyClass.changer(anObject); is executed and so the following are executed:
  aParameter.name = "Hot Shot";
  aParameter.number = 42;
As a result, anObject is changed.

Someplace else in memory:
Differences Between Primitive and Class-Type Parameters

• A method cannot change the value of a variable of a primitive type that is an argument to method.

• A method can change the values of the instance variables of a class type that is an argument to method.
public class ToyClass2 {
    private String name;
    private int number;

    public void set(String newName, int newNumber) {
        name = newName;
        number = newNumber;
    }

    public String toString() {
        return (name + " " + number);
    }
}
public void makeEqual(ToyClass2 anObject)
{
    anObject.name = this.name;
    anObject.number = this.number;
}

public void tryToMakeEqual(int aNumber)
{
    aNumber = this.number;
}

public boolean equals(ToyClass2 otherObject)
{
    return ( (name.equals(otherObject.name))
            && (number == otherObject.number) );
}

<Other methods can be the same as in Display 5.11, although no other methods are needed or used in the current discussion.>
public class ParametersDemo
{
    public static void main(String[] args)
    {
        ToyClass2 object1 = new ToyClass2(),
        object2 = new ToyClass2();
        object1.set("Scorpius", 1);
        object2.set("John Crichton", 2);
        System.out.println("Value of object2 before call to method:");
        System.out.println(object2);
        object1.makeEqual(object2);
        System.out.println("Value of object2 after call to method:");
        System.out.println(object2);

        int aNumber = 42;
        System.out.println("Value of aNumber before call to method: " + aNumber);
        object1.tryToMakeEqual(aNumber);
        System.out.println("Value of aNumber after call to method: " + aNumber);
    }
}
Comparing Parameters of a Class Type and a Primitive Type (Part 2 of 2)

Sample Dialogue

Value of object2 before call to method:
John Crichton 2

Value of object2 after call to method:
Scorpius 1

Value of aNumber before call to method: 42

Value of aNumber after call to method: 42

An argument of a class type can change.

An argument of a primitive type cannot change.
Pitfall: Use of = and == with Variables of a Class Type

• Used with variables of a class type, the assignment operator (==) produces two variables that name the same object
  – This is very different from how it behaves with primitive type variables

• The test for equality (==) also behaves differently for class type variables
  – The == operator only checks that two class type variables have the same memory address
  – Unlike the equals method, it does not check that their instance variables have the same values
  – Two objects in two different locations whose instance variables have exactly the same values would still test as being "not equal"
public class ToyClass2 {

    private String name;
    private int number;

    public void set(String newName, int newNumber) {
        name = newName;
        number = newNumber;
    }

    public boolean equals(ToyClass2 otherObject) {
        return (name.equals(otherObject.name))
            && (number == otherObject.number);
    }
}
ToyClass2 variable1 = new ToyClass2("Chiana", 3),
        variable2 = new ToyClass2("Chiana", 3);
if (variable1 == variable2)
    System.out.println("Equal using ==");
else
    System.out.println("Not equal using ==");

This code will produce the output

Not equal using ==
The Constant **null**

- **null** is a special constant that may be assigned to a variable of any class type
  
  ```java
  YourClass yourObject = null;
  ```

- It is used to indicate that the variable has no "real value"
  - It is often used in constructors to initialize class type instance variables when there is no obvious object to use

- **null** is not an object: It is, rather, a kind of "placeholder" for a reference that does not name any memory location
  - Because it is like a memory address, use `==` or `!=` (instead of `equals`) to test if a class variable contains null
    
    ```java
    if (yourObject == null) . . .
    ```
Pitfall: Null Pointer Exception

• Even though a class variable can be initialized to null, this does not mean that null is an object
  – null is only a placeholder for an object

• A method cannot be invoked using a variable that is initialized to null
  – The calling object that must invoke a method does not exist

• Any attempt to do this will result in a "Null Pointer Exception" error message
  – For example, if the class variable has not been initialized at all (and is not assigned to null), the results will be the same
Material Covered

• Chapter 5: pages 317 – 324
  – Absolute Java, 4th edition, Savitch
Source

• Absolute Java, Savitch
  - Copyright © 2010 Pearson Addison-Wesley. All rights reserved.