



Object-Oriented Programming

Lab #4, #5

● Contents

- Static In Java
- Math Class
- Wrapper Classes
- Variables and Memory
- Class Parameters
- new
- null
- privacy leaks
- copy constructor
- deep copy vs. shallow copy

CSLAB

● Class Outline

```
public class ClassName{  
    instance variables;  
    Constructors(){...}  
    getters(){...} //accessors  
    setters(parameters){...} //mutators  
    other methods(){...}  
    equals(){...}  
    toString(){...}  
}
```

CSLAB

● Class Outline (Contd)

Display 5.11 A Simple Class

```
1  public class ToyClass
2  {
3      private String name;
4      private int number;
5
6      public ToyClass(String initialName, int initialNumber)
7      {
8          name = initialName;
9          number = initialNumber;
10
11      public ToyClass()
12      {
13          name = "No name yet.";
14          number = 0;
15
16      public void set(String newName, int newNumber)
17      {
18          name = newName;
19          number = newNumber;
20
21      public String toString()
22      {
23          return (name + " " + number);
24
25      public static void changer(ToyClass aParameter)
26      {
27          aParameter.name = "Hot Shot";
28          aParameter.number = 42;
29
30      public boolean equals(ToyClass otherObject)
31      {
32          return ((name.equals(otherObject.name))
33                  && (number == otherObject.number));
34      }
```

instance variables

constructors

setter (mutator)

toString()

other methods

equals()

CSLAB

● Static

- A class can have data and methods of its own (not part of the objects)
- For example, *Employee* class can keep a count of the number of objects it has created

```
class Employee{  
    public static int empCount; //not an instance variable  
    ...  
    Employee(String name, int age, int salary){  
        this.name = name;  
        this.age = age;  
        this.salary = salary;  
        empCount++; //increment the number of employees  
    }  
}
```

CSLAB

● Static (Contd)

• In Java

```
public class EmployeeManager {  
    public static void main (String[] args){  
        Employee emp1 = new Employee("James Wright",42,"Manager", 20000);  
  
        Employee emp2 = new Employee("Amy Smith",27,"Design Coordinator", 8000,15);  
        Employee emp3 = new Employee("Peter Coolidge",32,"Assistant Manager", 12000,7);  
        Employee emp4 = new Employee("John Doe",22,"Engineer", 10000,10);  
  
        System.out.println(emp1.toString()+emp2.toString()+emp3.toString()+emp4.toString());  
        System.out.println("Number of Employees: "+Employee.empCount +"\n");  
    }  
}
```

- **Employee.empCount** is asking Employee class for the count of the employees created
 - Since empCount belongs to class and not the object
 - Hence use the class name (Employee) and not object name (emp1,2,3,4) to access variable empCount

CSLAB

• Static (Contd)

- Output after running the program.

```
My Name is: James Wright
My Age is: 42
My Position is: Manager
My Salary is: 20000
My vacation days: 20

My Name is: Amy Smith
My Age is: 27
My Position is: Design Coordinator
My Salary is: 8000
My vacation days: 15

My Name is: Peter Coolidge
My Age is: 32
My Position is: Assistant Manager
My Salary is: 12000
My vacation days: 7

My Name is: John Doe
My Age is: 22
My Position is: Engineer
My Salary is: 10000
My vacation days: 10

Number of Employees: 4
```

CSLAB

● Static (Contd)

- In Java it is possible to declare the following as static
 - Variables
 - e.g. `static int empCount;`
 - Methods
 - e.g.

```
public static double area(double radius){  
    //calculates the area given the radius  
}
```
 - Classes
 - Usually used with static inner classes.
 - More on this in chap 13
 - Static Blocks
 - e.g.

```
static {  
    ...  
}
```

CSLAB

• Static Variables

- A static variable is a variable that belongs to the class as a whole, and not just to one object
 - There is only **one copy of a static variable per class**, unlike instance variables where each object has its own copy
- All objects of the class can read and change a static variable
- Although a static method cannot access an instance variable, a static method can access a static variable
- A static variable is declared like an instance variable, with the addition of the modifier *static*
 - *private static int myStaticVariable;*

CSLAB

● Static Variables (Contd)

- Static variable should be defined as *private*
 - *private static int empCount;*
- unless you are defining a constant.
 - *public static final double PI = 3.14159;*

CSLAB

● Static Methods

- A static method is one that can be used *without a calling object*
- A static method still belongs to a class, and its definition is given inside the class definition
- When a static method is defined, the keyword *static* is placed in the method header
 - *public static returnType myMethod(parameters) { ... }*
- Static methods are invoked using the class name in place of a calling object
 - *returnValue = MyClass.myMethod(arguments);*

CSLAB

Static Methods (Contd)

```
/**
 * Class with static methods for circles and spheres.
 */
public class RoundStuff
{
    public static final double PI = 3.14159;

    /**
     * Return the area of a circle of the given radius.
     */
    public static double area(double radius)
    {
        return (PI*radius*radius);
    }

    /**
     * Return the volume of a sphere of the given radius.
     */
    public static double volume(double radius)
    {
        return ((4.0/3.0)*PI*radius*radius*radius);
    }
}
```

```
import java.util.Scanner;

public class RoundStuffDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter radius:");
        double radius = keyboard.nextDouble();

        System.out.println("A circle of radius "
            + radius + " inches");
        System.out.println("has an area of " +
            RoundStuff.area(radius) + " square inches.");
        System.out.println("A sphere of radius "
            + radius + " inches");
        System.out.println("has an volume of " +
            RoundStuff.volume(radius) + " cubic inches.");
    }
}
```

CSLAB

● Main

- The *Main* method is a well known static method
- You can put a *main* in any class
- Although the main method is often by itself in a class separate from the other classes of a program, it can also be contained within a regular class definition
 - In this way the class in which it is contained can be used to create objects in other classes, or it can be run as a program
 - A main method so included in a regular class definition is especially useful when it contains diagnostic code for the class

CSLAB

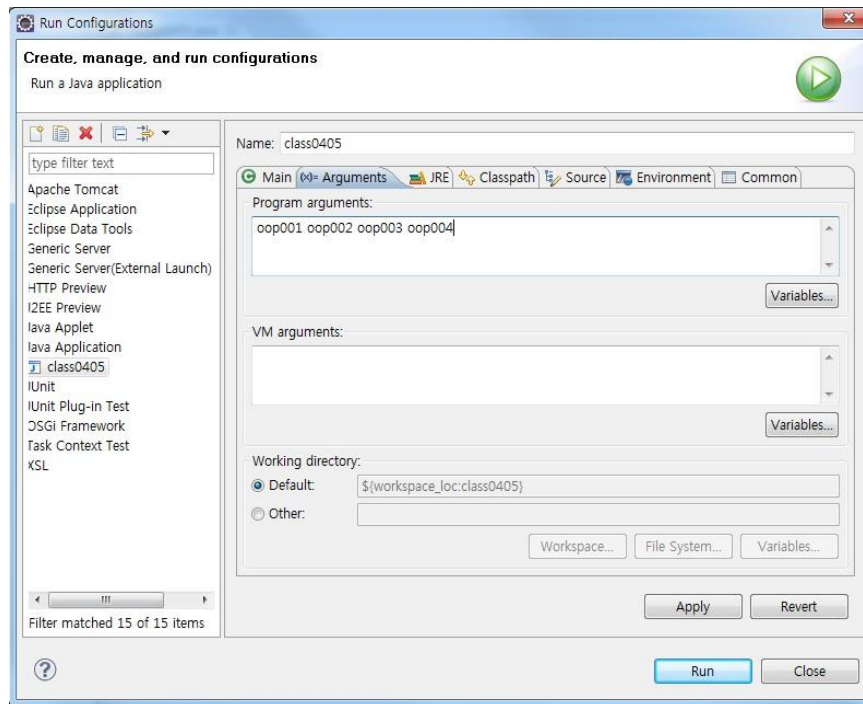
• String args[]

- Parameter in the main method
- Command-line arguments in Java
- When an application is launched, the runtime system passes the command-line arguments to the application's main method via an array of Strings.

CSLAB

• String args[]

- From the menu bar select Run -> Run Configuration



```
public class class0405{  
  
    public static void main(String[] args)  
    {  
        int i;  
        for(i=0;i<args.length;i++)  
            System.out.println(args[i]);  
    }  
}
```

The screenshot shows the Eclipse IDE console. The output of the Java application is displayed, showing the sequence of arguments passed to the main method: 'oop001', 'oop002', 'oop003', and 'oop004'. The console title bar indicates the application is 'class0405 [Java Application]' and the working directory is 'C:\Program Files\Java\'. The console also shows a 'terminated' status.

CSLAB

● The Math Class

- The Math class is an important built-in class
- Contains static variables and methods.
- Collection of common math functions (sin, cos, sqrt, etc.)
- Two constants: PI and E

CSLAB

● The Math Class (Contd)

- The *Math* class provides a number of standard mathematical methods
 - It is found in the *java.lang* package, so it does not require an *import* statement
 - All of its methods and data are static, therefore they are invoked with the class name *Math* instead of a calling object
 - The *Math* class has two predefined constants, *E* (e, the base of the natural logarithm system) and *PI* (π , 3.14159 ...)
 - *area = Math.PI * radius * radius;*

CSLAB

● Structure of the Math Class

```
public class Math {  
    public static final double PI = 3.141592653589793;  
  
    public static double sin(double d) { ... }  
    public static double sqrt(double d) { ... }  
  
    private Math() {}  
    ...  
}
```

CSLAB

● What's different about Math Class

- It's different from a typical Java class

- It is a "stateless" class
- We only need one Math class (not multiple instances)
- No need to instantiate it (hence, no public constructor)
- All of its variables and methods are *static*
- *static* means "applies to the class as a whole" vs. "applies to an individual instance"

CSLAB

Some Methods in the class *Math*

(Part 1 of 5)

Display 5.6 Some Methods in the Class Math

The Math class is in the `java.lang` package, so it requires no `import` statement.

```
public static double pow(double base, double exponent)
```

Returns base to the power exponent.

EXAMPLE

`Math.pow(2.0, 3.0)` returns `8.0`.

(continued)

CSLAB

Some Methods in the class *Math*

(Part 2 of 5)

Display 5.6 Some Methods in the Class Math

```
public static double abs(double argument)
public static float abs(float argument)
public static long abs(long argument)
public static int abs(int argument)
```

Returns the absolute value of the argument. (The method name `abs` is overloaded to produce four similar methods.)

EXAMPLE

`Math.abs(-6)` and `Math.abs(6)` both return 6. `Math.abs(-5.5)` and `Math.abs(5.5)` both return 5.5.

```
public static double min(double n1, double n2)
public static float min(float n1, float n2)
public static long min(long n1, long n2)
public static int min(int n1, int n2)
```

Returns the minimum of the arguments `n1` and `n2`. (The method name `min` is overloaded to produce four similar methods.)

EXAMPLE

`Math.min(3, 2)` returns 2.

(continued)

Some Methods in the class *Math*

(Part 3 of 5)

Display 5.6 Some Methods in the Class Math

```
public static double max(double n1, double n2)
public static float max(float n1, float n2)
public static long max(long n1, long n2)
public static int max(int n1, int n2)
```

Returns the maximum of the arguments n1 and n2. (The method name max is overloaded to produce four similar methods.)

EXAMPLE

`Math.max(3, 2)` returns 3.

```
public static long round(double argument)
public static int round(float argument)
```

Rounds its argument.

EXAMPLE

`Math.round(3.2)` returns 3; `Math.round(3.6)` returns 4.

(continued)



Some Methods in the class *Math*

(Part 4 of 5)

Display 5.6 Some Methods in the Class Math

```
public static double ceil(double argument)
```

Returns the smallest whole number greater than or equal to the argument.

EXAMPLE

`Math.ceil(3.2)` and `Math.ceil(3.9)` both return 4.0.

(continued)

CSLAB

Some Methods in the class *Math*

(Part 5 of 5)

Display 5.6 Some Methods in the Class Math

```
public static double floor(double argument)
```

Returns the largest whole number less than or equal to the argument.

EXAMPLE

`Math.floor(3.2)` and `Math.floor(3.9)` both return `3.0`.

```
public static double sqrt(double argument)
```

Returns the square root of its argument.

EXAMPLE

`Math.sqrt(4)` returns `2.0`.

CSLAB

● Random Numbers

- The Math class also provides a facility to generate pseudo-random numbers

public static double random()

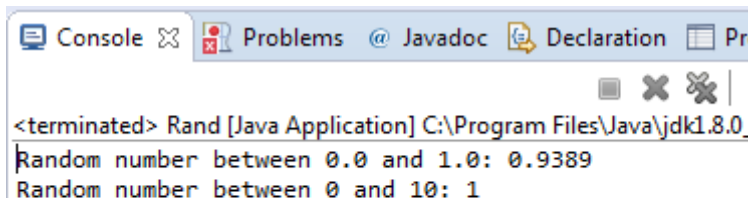
- A pseudo-random number appears random but is really generated by a deterministic function
 - There is also a more flexible class named Random
- Sample use: *double num = Math.random();*
- Returns a pseudo-random number greater than or equal to 0.0 and less than 1.0

CSLAB

• Random Numbers

• Example

```
public class Rand {  
    public static void main(String[] args){  
        //Print a random number between 0.0 and 1.0  
        System.out.printf("Random number between 0.0 and 1.0: %3.4f \n",Math.random());  
  
        //Print a random number between 0 and 10  
        System.out.printf("Random number between 0 and 10: %d \n",(int)(Math.random()*10));  
    }  
}
```



The screenshot shows a Java IDE window with tabs for Console, Problems, Javadoc, Declaration, and Properties. The Console tab is active, displaying the output of the program. The output shows two lines: "Random number between 0.0 and 1.0: 0.9389" and "Random number between 0 and 10: 1".

```
<terminated> Rand [Java Application] C:\Program Files\Java\jdk1.8.0_...  
Random number between 0.0 and 1.0: 0.9389  
Random number between 0 and 10: 1
```

CSLAB

● Self-Test

- `int a = 60984, int b = 808` 일 때, `Math` 클래스의 2가지 메소드를 호출하여 두 숫자 중 더 큰 숫자와, 더 작은 숫자를 출력할 것
- `double x = 2.0, double y = 3.0` 일 때, `Math` 클래스의 메소드를 호출하여 x^y (x 의 y 승)과 y^x (y 의 x 승)을 출력할 것

CSLAB

● Primitives & Wrapper Classes

- Java's primitive data types (boolean, int, etc.) are not classes.
- Wrapper classes are used in situations where objects are required.

CSLAB

● Primitives & Wrapper Classes (Contd)

- Java has a wrapper class for each of the eight primitive data types:

Primitive Type	Wrapper Class	Primitive Type	Wrapper Class
boolean	Boolean	float	Float
byte	Byte	int	Integer
char	Character	long	Long
double	Double	short	Short

CSLAB

● Boxing & Unboxing

- Boxing is the process of wrapping a primitive in its wrapper class object.
- Simply use the constructor and pass the primitive as a parameter
 - *Integer integerObj = new Integer(42);*
 - *Double doubleObj = new Double(3.2);*
 - *Boolean boolObj = new Boolean(true);*
- *Automatic boxing is possible*
 - *Integer integerObj = 42;*
 - *Double doubleObj = 3.2;*
 - *Boolean boolObj = true;*

CSLAB

● Boxing & Unboxing (Contd)

- Unboxing is the process of unwrapping a primitive from its wrapper class object
- Simply use the `<typename>Value()` method
 - `int i = integerObj.intValue();`
 - `double d = doubleObj.doubleValue();`
 - `boolean b = booleanObj.booleanValue();`
- Automatic Unboxing is possible
 - `int i = integerObj;`
 - `double d = doubleObj;`
 - `boolean b = booleanObj;`

CSLAB

• Strings & Wrappers

- The Wrapper class for each primitive *type* has a method `parseType()` to parse a string representation & return the literal value.
 - `Integer.parseInt("42")` $\Rightarrow 42$
 - `Boolean.parseBoolean("true")` $\Rightarrow true$
 - `Double.parseDouble("2.71")` $\Rightarrow 2.71$
- *Common use: Parsing the arguments to a program*

CSLAB

● Parsing arguments lists

- This method accepts an array of parameters

```
5 public void parseArgs(String[] args) {  
6     for(int i = 0; i < args.length; i++) {  
7         try {  
8             System.out.println(Integer.parseInt(args[i]));  
9         } catch (Exception e) {  
10            try {  
11                System.out.println(Float.parseFloat(args[i]));  
12            } finally { }  
13        }  
14    }  
15 }
```

arg # 0 = 0
arg # 1 = 42
arg # 2 = 999
arg # 3 = 0.0
arg # 4 = 1.42
arg # 5 = 9.0008

CS LAB

● Strings & Wrappers

- Wrapper classes also have static methods that convert from a numeric value to a string representation of the value
- For example, the expression
 - *Double.toString(123.99);*
 - returns the string value "123.99"

CSLAB

● Example

```
int i = 1234;  
String str = Integer.toString(i);  
  
double total = 44;  
String total2 = Double.toString(total);  
  
String str = "1234";  
int num = Integer.parseInt(str);
```

CSLAB

● Wrapper class

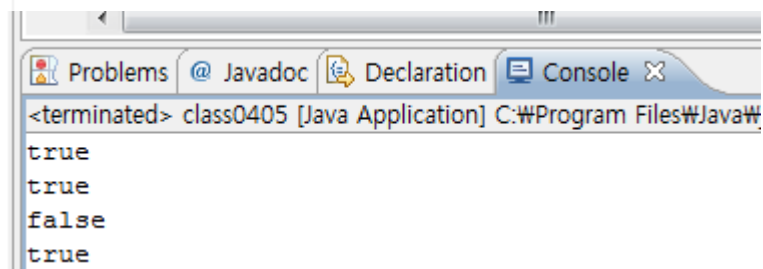
```
import java.lang.Math;

public class class0405{

    public static void main(String[] args)
    {
        int a = 1;
        int b = 1;

        Integer c = new Integer(1);
        Integer d = 1;

        System.out.println(a==b);
        System.out.println(a==c);
        System.out.println(c==d);
        System.out.println(c.equals(d));
    }
}
```



The screenshot shows a Java IDE window with a console tab. The console output is as follows:

```
<terminated> class0405 [Java Application] C:\Program Files\Java\
true
true
false
true
```

- **NOTE:** The operator `==` checks equality of memory address, it should not be used to compare 2 objects.
- You should use the `equals()` to compare 2 objects

CSLAB

● Wrapper class (Contd)

- For each number of Wrapper has a MAX_VALUE constant:

```
byteObj = new Byte(Byte.MAX_VALUE);  
shortObj = new Short(Short.MAX_VALUE);  
intObj = new Integer(Integer.MAX_VALUE);  
longObj = new Long(Long.MAX_VALUE);  
floatObj = new Float(Float.MAX_VALUE);  
doubleObj = new Double(Double.MAX_VALUE);  
  
printNumValues("MAXIMUM NUMBER VALUES:");
```

=>

```
Byte:127  
Short:32767  
Integer:2147483647  
Long:9223372036854775807  
Float:3.4028235E38  
Double:1.7976931348623157E308
```

CSLAB

● Wrapper class (Contd)

- Many useful utility methods, e.g. for Integer:

```
int hashCode()  
static int numberOfLeadingZeros(int i)  
static int numberOfTrailingZeros(int i)  
static int reverse(int i)  
static int reverseBytes(int i)  
static int rotateLeft(int i, int distance)  
static int rotateRight(int i, int distance)  
static String toBinaryString(int i)  
static String toHexString(int i)  
static String toOctalString(int i)  
static String toString(int i, int radix)
```

CSLAB

● Wrapper class (Contd)

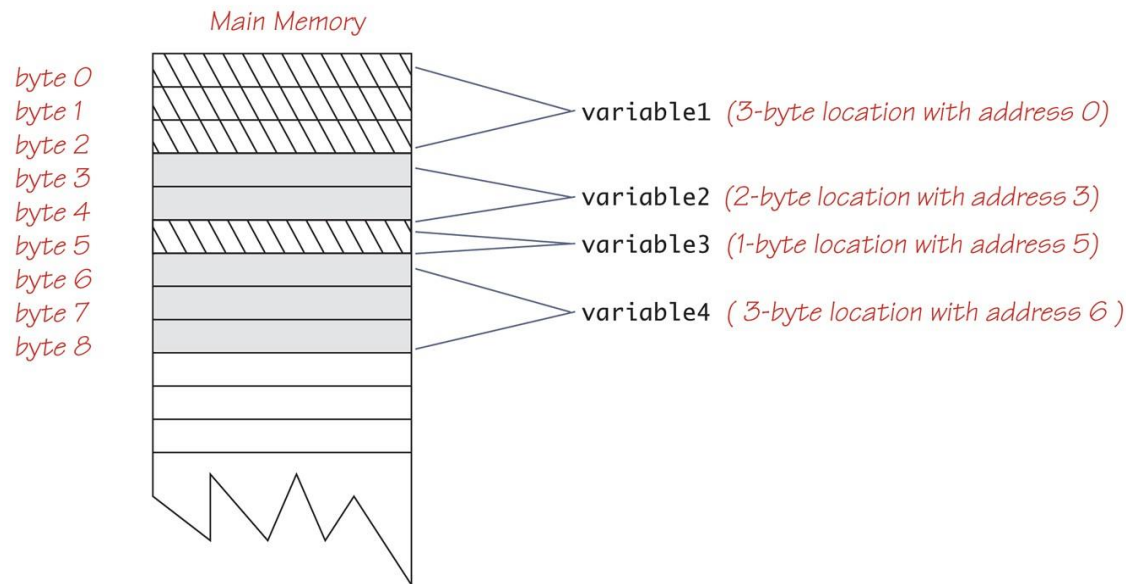
- **Double & Float: Utilities for arithmetic operations**
 - Constants: POSITIVE_INFINITY & NEGATIVE_INFINITY
 - Constants: NaN = Not a Number value
 - *Methods is NaN(), isInfinite()*

CSLAB

Variables and Memory

- Main memory is used to store data in contiguous bytes.
- The address of the first byte is the address of the data item.

Display 5.10 Variables in Memory



CSLAB

● Variables and Memory

- Primitive types store the actual value at the memory location

- `int i = 20;`
- `char c = 'h';`
- `short sh = 10;`

0	20
1	
2	
3	
4	h
5	
6	10
7	
8	
9	
10	

CSLAB

● References

- The memory address to a class variable is its reference
- References are treated differently since the object is not directly stored here
- References store the memory address where the object is actually stored

CSLAB

References (Contd)

Display 5.12 Class Type Variables Store a Reference

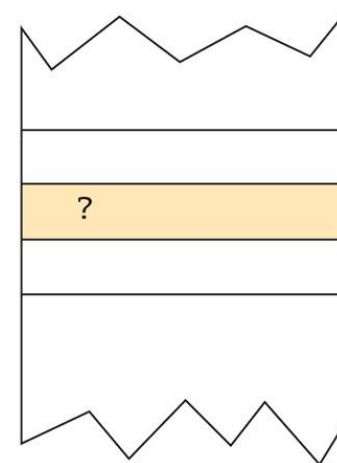
```
public class ToyClass
{
    private String name;
    private int number;
```

The complete definition of the class ToyClass is given in Display 5.11.

```
ToyClass sampleVariable;
```

*Creates the variable **sampleVariable** in memory but assigns it no value.*

sampleVariable



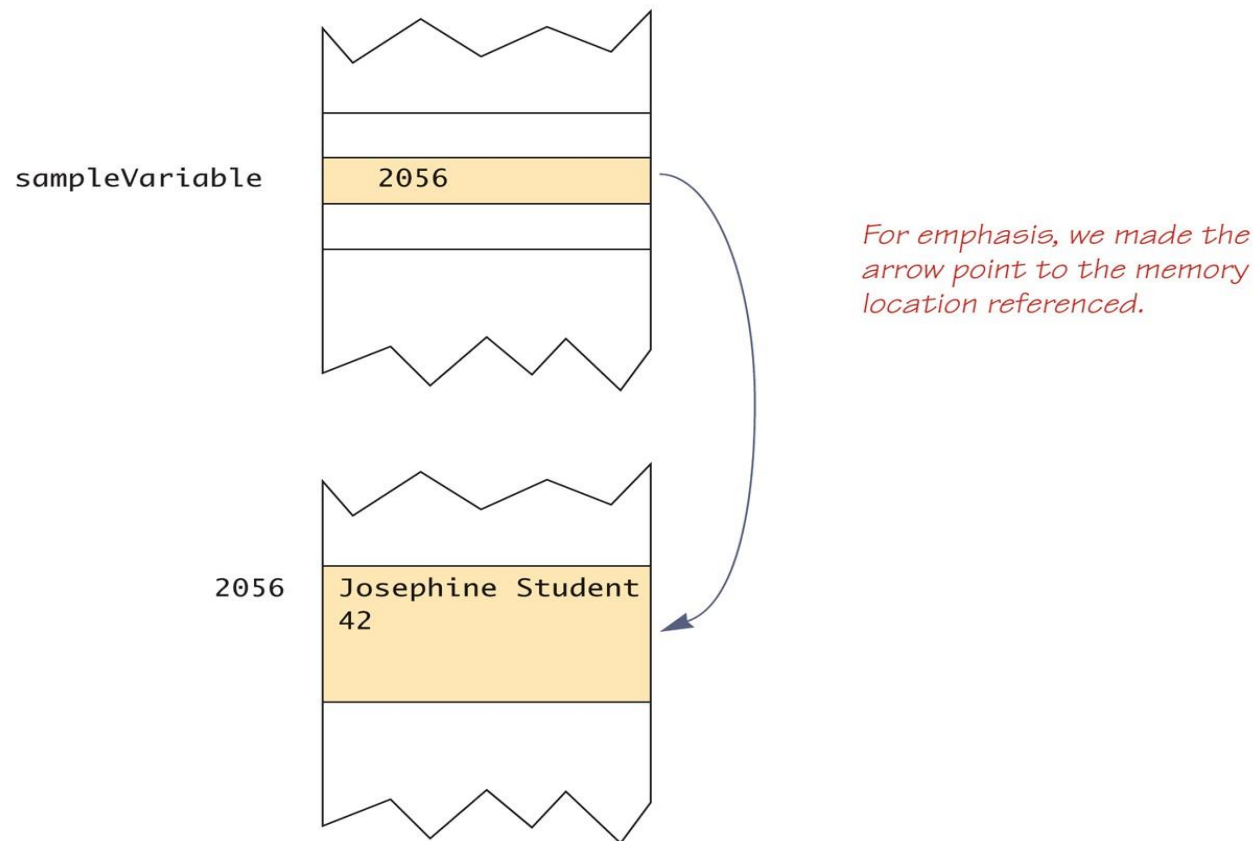
```
sampleVariable =
new ToyClass("Josephine Student", 42);
```

*Creates an object, places the object someplace in memory, and then places the address of the object in the variable **sampleVariable**. We do not know what the address of the object is, but let's assume it is 2056. The exact number does not matter.*

(continued)

References (Contd)

Display 5.12 Class Type Variables Store a Reference



CSLAB

● Class Parameters

• Call-by-Value

- Java uses the call-by-value mechanism for parameters of primitive-type
- The actual parameter (or argument expression) is fully evaluated and the resulting value is *copied* into a location being used to hold the formal parameter's value during method execution
- That location is typically a chunk of memory on the runtime stack for the application

CSLAB

● Class Parameters (Contd)

```
package test;

public class Test {

    public static void swap(int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }

    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        System.out.println("swap() 메소드 호출 <전>: a="+a+", b="+b);
        swap(a,b);
        System.out.println("swap() 메소드 호출 <후>: a="+a+", b="+b);
    }

}
```

- The values of A and B have not been changed!!!

<terminated> Test [Java Application] C:₩

```
swap() 메소드 호출 <전>: a=10, b=20
swap() 메소드 호출 <후>: a=10, b=20
|
```

CSLAB

● Class Parameters (Contd)

- In case the parameter is of a class type, the value plugged in is a reference (memory address).
- Any change made to the parameter is made to the object named by the parameter, because they are the same object. Thus, a method can change the instance variables of an object given as a parameter.

CSLAB

● Class Parameters (Contd)

```
package test;

class Number {
    public int a;
    public int b;
}

public class Test {

    public static void swap(Number z) {
        int temp = z.a;
        z.a = z.b;
        z.b = temp;
    }

    public static void main(String[] args) {

        Number n = new Number();
        n.a = 10;
        n.b = 20;
        System.out.println("swap() 메소드 호출 <전>: a="+n.a+", b="+n.b);
        swap(n);
        System.out.println("swap() 메소드 호출 <후>: a="+n.a+", b="+n.b);

    }

}
```

<terminated> Test [Java Application] C:\WP\

swap() 메소드 호출 <전>: a=10, b=20
swap() 메소드 호출 <후>: a=20, b=10

- In short: A method cannot change the value of a parameter of a primitive type. But a method can change the value of an object

CSLAB

● Classes: Review

- **Static methods**
 - Can be used without a calling object
- **Static variables**
 - One copy of a static variable per class
- **The Math class**
 - Provides many standard mathematical methods
- **Wrapper classes**
 - Class types corresponding to primitive types
- **Passing Parameters**
 - A method cannot change the value of a parameter of a primitive type.
But a method can change the value of an object

CSLAB

● null

- null is a special constant that may be assigned to a variable of any class type

YourClass yourObject = null;



\$null\$

- 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
 - *if (yourObject == null)*

CSLAB

● new

- The *new* operator invokes a constructor which initializes an object, and returns a reference to the location in memory of the object created
 - This reference can be assigned to a variable of the object's class type
- An object whose reference is not assigned to a variable is called an *anonymous object*

```
if (variable1.equals(new ToyClass("Joe", 42)))  
    System.out.println("Equal");  
else  
    System.out.println("Not equal");
```

CSLAB

● Maintaining Class privacy

- For a primitive type instance variable, just adding the *private* modifier to its declaration should insure that there will be no privacy leaks
- For a *class type* instance variable, however, adding the *private* modifier alone is *not sufficient*

For example:

```
public Date getBirthDate()  
{  
    return born; //dangerous  
}
```

- Why is this dangerous?

CSLAB

● Copy Constructors

- A *copy constructor* is a constructor with a single argument of the same type as the class
- The copy constructor should create object that is a *separate, independent object*, but with the instance variables set so that it is an *exact copy* of the argument object

CSLAB

Copy Constructor for a class with Primitive Type Instance Variables

```
public Date (Date aDate)  
{  
    if (aDate == null) // Not a real date  
    {  
        System.out.println("Fatal Error. ");  
        System.exit(0);  
    }  
  
    month = aDate.month;  
    day = aDate.day;  
    year = aDate.year;  
}
```

CSLAB

Copy Constructor (Contd)

```
package test;

class person{

    private String name;
    private String address;

    public person(String setName,String setAddress) {
        // TODO Auto-generated constructor stub
        this.set(setName, setAddress);
    }
    public person(person p){

        if(p==null)
        {
            System.out.println("Fatal Error");
            System.exit(0);
        }

        name = p.name;
        address = p.address;
    }

    void set(String setName, String setAddress){

        this.name = setName;
        this.address = setAddress;
    }
    void display(){

        System.out.println("Name : "+this.name+" Address : "+this.address);
    }
}
```

```
public class Test {

    public static void main(String arg[])
    {
        person person1= new person("kim", "Seoul");
        person person2; //쓰레기값으로 초기화

        person person3 = person1;
        person person4 = new person(person1);

        person1.display();
        person3.display();
        person4.display();

        person1.set("lee", "Busan");

        person1.display();
        person3.display();
        person4.display();
    }
}
```

<terminated> Test [Java Application]

```
Name : kim Address : Seoul
Name : kim Address : Seoul
Name : kim Address : Seoul
Name : lee Address : Busan
Name : lee Address : Busan
Name : kim Address : Seoul
```

- Notice the difference between person 3 and 4

CSLAB

● Deep Copy vs. Shallow Copy

- A *deep copy* of an object is a copy that, with one exception, has no references in common with the original
 - Exception: References to immutable objects are allowed to be shared
- Any copy that is not a deep copy is called a *shallow copy*
 - This type of copy can cause dangerous privacy leaks in program

CSLAB

● Deep Copy vs. Shallow Copy (Contd)

```
import java.util.Arrays;

public class Test {

    public static void main(String[] args) {
        int[] data = {0,1,2,3,4};
        int[] sCopy = null;
        int[] dCopy = null;

        sCopy = shallowCopy(data);
        dCopy = deepCopy(data);

        System.out.println("Original:" + Arrays.toString(data));
        System.out.println("Shallow :" + Arrays.toString(sCopy));
        System.out.println("Deep      :" + Arrays.toString(dCopy));
        System.out.println();

        /* 원본 데이터 변경 */
        data[0] = 5;
        System.out.println("Original:" + Arrays.toString(data));
        System.out.println("Shallow :" + Arrays.toString(sCopy));
        System.out.println("Deep      :" + Arrays.toString(dCopy));
    }
}
```

CSLAB

● Deep Copy vs. Shallow Copy (Contd)

```
public static int[] shallowCopy(int[] arr) {  
    return arr;  
}  
  
public static int[] deepCopy(int[] arr) {  
    if(arr==null) return null;  
    int[] result = new int[arr.length];  
  
    System.arraycopy(arr, 0, result, 0, arr.length);  
    return result;  
}
```

```
<terminated> Test [Java Application] C  
Original:[0, 1, 2, 3, 4]  
Shallow :[0, 1, 2, 3, 4]  
Deep    :[0, 1, 2, 3, 4]  
  
Original:[5, 1, 2, 3, 4]  
Shallow :[5, 1, 2, 3, 4]  
Deep    :[0, 1, 2, 3, 4]
```

CSLAB

• Mutable vs. Immutable Classes

- When creating classes we create accessors and mutators to allow us to manipulate private instance variables.
- Classes such as this are said to be *mutable*
 - i.e. It is possible to change the values of instance variables after construction.
- Having Mutators in classes may break the rules for avoiding privacy leaks

CSLAB

● Mutable vs. Immutable Classes (Contd)

- A class that contains no methods (other than constructors) that change any of the data in object of the class is called an *immutable class*
 - Objects of such a class are called *immutable objects*
 - It is perfectly safe to return a reference to an immutable object because the object cannot be changed in any way
 - The *String*, *Math* class is an immutable class

CSLAB

● Self-Test

- String name, double lat, double lon을 instance variable로 가지는 City 클래스를 생성한다. (modifier: private)
- 모든 instance variable의 값을 설정하는 생성자를 만든다.
- name instance variable만 설정하는 생선자를 만든다.
 - 이 경우 lat과 lon은 0이상 360미만의 난수값이 된다.
- 해당 City가 다른 City와 동일한지 검사하는 equals() 메소드를 만든다.
 - City의 name, lat, lon이 모두 같아야 동일한 City이다.
- City의 name과 lat과 lon을 문자열로 한 문장에 반환하는 toString() 메소드를 만든다.
- 두 도시의 거리를 계산하는 cityDistance static method를 만든다. cityDistance 메소드를 호출할 때는 반드시 메소드명이 아닌 클래스명을 사용하여 호출해야 한다.

$$distance = \sqrt{(lon1 - lon2)^2 + (lat1 - lat2)^2}$$

CSLAB

● Self-Test (Contd)

- Cities 클래스에 생성되어 있는 객체를 이용하여 다음과 같은 내용을 출력할 것
 - 도시의 name, lat, lon을 한 문장에 출력하라: *Seoul, Mega City*
 - 도시 간의 동일 여부를 출력하라: *Paris - Paris, Seoul - Paris*
 - 도시 간의 거리를 출력하라: *Seoul - Paris, Seoul - Racoon City, Paris - Mega City*
 - cityDistance 메소드를 호출할 때는 반드시 메소드명이 아닌 클래스명을 사용하여 호출해야 한다.

```
Seoul Information: Seoul, 344.4291482082959, 357.00411896035735  
Mega City Information: Mega City, 310.0, 170.0
```

```
Paris-Paris: true  
Seoul-Paris: false
```

```
Seoul-Pris: 236.51460409168635  
Seoul-Racoon City: 332.13787907656183  
Paris-Mega City: 194.65353837010002
```

CSLAB