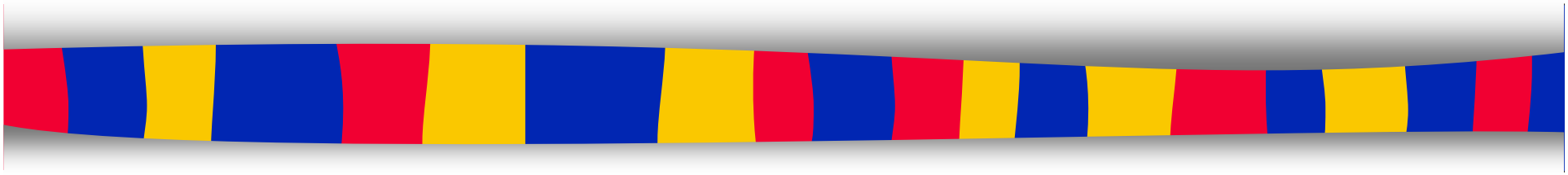
COMP-248
Object Oriented Programming I



Defining Classes II

By Emad Shihab, PhD, Fall 2015,
Parts of the slides are taken from Prof. L. Kosseim
Adapted for Section EE by S. Ghaderpanah, Fall 2015

Next:

1. **`static`** methods and variables
2. Wrapper classes
3. References and class parameters
4. Using and Misusing references
5. Packages and **`javadoc`**

Static Members

Members can be:

- public / private / protected / package
- static (class members) / non-static (instance members)

Instance members (non **`static`**)

- associates a variable or method with **each object**
- invoked through the name of a **specific object**

```
myAccount.deposit(10);
system.out.print(yourCoin.face);
```

Class members (**`static`**)

- associates a variable or method with **each class** (shared by all objects of the class)
- invoked through the **name of the class**

```
System.out.print(Math.sqrt(25));
if (Character.isUpperCase('a'))
   …
```

Static Methods

Static methods (class methods)

- special **methods that pertain to the class**… not to a specific object
- are **called through the name of the class**… not through an object

```
System.out.print(Math.max(3, 10));
System.out.print(Math.sqrt(9));
System.out.print(Math.random());
```

Non-static methods (instance methods)

- methods that pertain to a **specific object**
- are called through the **name of a specific object**

```
Random gen = new Random();
int num1 = gen.nextInt();
```

calling object

```
String aNoun = "Mary";
String aVerb = "eats";

if (aNoun.length() > 10)
    …
System.out.print(aVerb.charAt(3));
```

Static Methods

```
public class RoundStuff
{
    public static final double PI = 3.14159;

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

Driver

```
…
double radius = keyboard.nextDouble( );
System.out.println(“The area is " +
RoundStuff.area(radius) + " square inches.");
System.out.println(”The volume is " +
RoundStuff.volume(radius) + " cubic inches.");
…
```

Static Methods

- An example of a static method is the **main** method
  - it is invoked by the system without creating an object
  - Static methods can be used without a calling object

- A static method **cannot refer to an instance variable** of the class, **nor invoke nonstatic methods** of the class (unless it creates a new object)
  - In other words: in the **definition of a static method**, you **cannot use an instance variable** or **method that has an implicit or explicit this** for a calling object

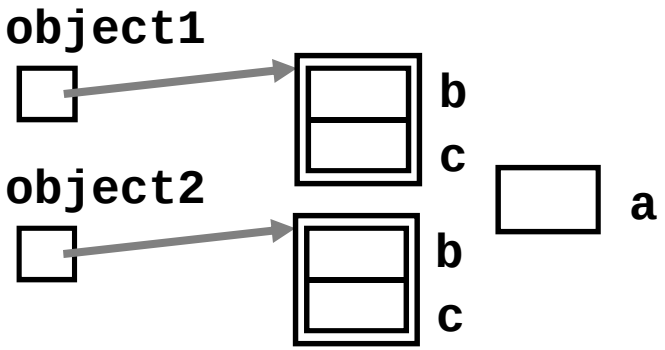
- Static methods are **called through the name of the class** (or even through an object)

Static Variables

- A static variable is a variable that belongs to the class as a whole
  - There is **only one copy of the variable and all objects** can use it
  - In a nonstatic variable each object has its own copy of the variable
- A static variable can be used to communicate between objects
  - For example: one object can change a static variable and another object can read it
- Can you access a static variable from a static method?

Static Variables

```
public class someClass {
   private static int a;
   private int b;
   private boolean c;
   …
}
```



Static Variables

- Can be initialized in the declaration
- If not explicitly initialized, it is automatically initialized to **`false / zero / null`**

```
public class SomeClass {
    private static int a = 10;
    private static int b;
    …
}
```

- space and initial value is assigned:
  - when the class is loaded for the 1st time, not every time an object is created
- constants are good candidates for being static variables

```
public static final int nbDays = 365;
```

- a static method:
  - cannot access an instance variable,
  - can access a static variable

Just Checking …

Only _____ copy/copies of a static variable are available to objects of a class.

A. zero
B. one
C. two
D. three
E. none of the above

Example

```
public class Account {
   private static int nbAccounts=0;
   private int balance;

   public Account() {
      nbAccounts++;
      balance = 0;
    }

    public static int getNbAccounts(){
      return nbAccounts; // 1. OK ?
    }

    public static int getBalance(){
      return balance; // 2. OK ?
    }
}
```

Account.java

```
System.out.print(Account.getNbAccounts());

Account myAccount = new Account();
System.out.print(myAccount.getNbAccounts());
```

Driver

Just Checking …

```
public class Account
{
  public static int nbAccounts=0;
  public int balance;

  public Account() {
    nbAccounts++;
    balance = 0;
  }
}
```

Account.java

```
Account a = new Account();
System.out.println(a.balance + " " + a.nbAccounts);

Account b = new Account();
System.out.println(a.balance + " " + a.nbAccounts);
System.out.println(b.balance + " " + b.nbAccounts);
```

Driver

A. 0 0
   0 0
   0 0

B. 0 1
   0 2
   0 3

C. 0 1
   0 2
   0 2

D. 0 2
   0 2
   0 2

E. Syntax error

Output

Example: java.lang.Math

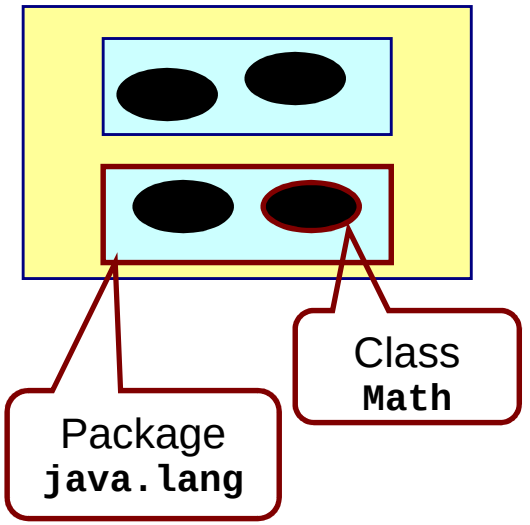
`java.lang.Math`

declares methods and math constants

constants: **`Math.PI, Math.E`**

methods: **`max(),min(),abs(),random(),sqrt(),…`**

all methods are static




```
Math.E      Math.max(3, 10)
Math.random()  Math.sqrt(9)
```

```
int radius;
double area, circumference;

System.out.print("Enter the circle's radius: ");
radius = keyboard.nextInt();

area = Math.PI * Math.pow(radius, 2);
circumference = 2 * Math.PI * radius;
```

In this chapter, we will see:

1. **`static`** methods and variables
2. **Wrapper classes**
3. References and class parameters
4. Using and Misusing references
5. Packages and **`javadoc`**

Wrapper Classes

- Java treats primitive types differently from class types
- Later on, we will see that **arguments to methods are treated differently**, depending on whether the argument is a **primitive type or class type**
- In some cases, we want to use a value of a primitive type, but need the value to be an object of a class type …this is where Wrapper classes become useful

2- Wrapper Classes

A wrapper class is an "container" to a primitive type

so we "wrap" the primitive type by some class

```
int x = 20;  // primitive type

Integer y = new Integer(20); // reference type
```

Wrapper classes

Every primitive type has a corresponding wrapper class in the **`java.lang`** package

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

- Wrapper classes also contain a number of useful predefined constants and static methods

Wrapper classes

Used for what?

wrapper classes contain:

many useful methods to help manage the associated type

ex: to convert an integer stored in a **`String`** to an **`int`** value:

```
String s = "2003";
int num = Integer.parseInt(str);
```

Some Methods in Wrapper Classes

- Methods to convert a string representation of a number to the number

  `Integer.parseInt`, **`Long.parseLong`**, **`Float.parseFloat`**, and **`Double.parseDouble`**

```
String s1;
s1 = keyboard.next();
double number1 = Double.parseDouble(s1);
```

- **`toString()`** methods to convert from a numeric value to a string representation of the value

```
Double.toString(123.99);
```

  returns the string value **`"123.99"`**

Boxing and Unboxing

- **Boxing**:

  converting a primitive value to an **object** of its wrapper class

```
Integer integerObject = new
  Integer(42);
```

- **Unboxing**:

  converting an object of a wrapper class to the corresponding value of a **primitive type**

  use the "typeValue()" methods…

```
int i = integerObject.intValue();

double d = doubleObject.doubleValue();

char c = charObject.charValue();

…
```

Automatic Boxing and Unboxing

Since Java 5.0, we have **automatic** boxing and unboxing

- Automatic boxing:
  Instead of creating a wrapper class object (as shown before)

```
Integer integerObject = new Integer(42);
```

  we can use a automatic type cast:

```
Integer integerObject = 42;
```

- Automatic unboxing:
  Instead of having to invoke the a method

```
int i = integerObject.intValue();
```

  the primitive value can be recovered automatically

```
int i = integerObject;
```

Just Checking …

All of the following are wrapper classes except:

A. `String`
B. `Integer`
C. `Character`
D. `Double`
E. `Boolean`

In this chapter, we will see:

1. **`static`** methods and variables
2. Wrapper classes
3. References and class parameters
4. Using and Misusing references
5. Packages and **`javadoc`**