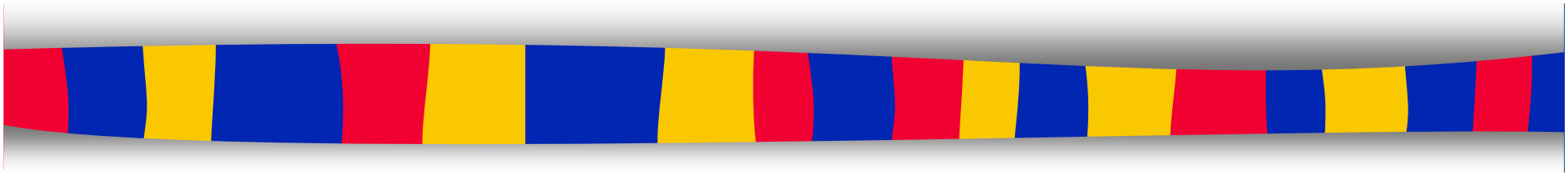
COMP-248
Object Oriented Programming I



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

Example 1: Swapping 2 Account

```
public static void main(String[] arg)
{
  Account a = new Account("ted", 123, 100);
  Account b = new Account("mary", 456, 99);

  System.out.println(a + " " + b);
  swap(a, b);
  System.out.println(a + " " + b);
}

public static void swap(Account c, Account d)
{
      Account tmp;
      tmp = c;
      c = d;
      d = tmp;
}
```

ted,123,100 mary,456,99
ted,123,100 mary,456,99

Output

Example 1: Swapping 2 Account

```
public static void main(String[] arg)
{
  Account a = new Account("ted", 123, 100);
  Account b = new Account("mary", 456, 99);

  System.out.println(a + " " + b);
  swap(a, b);
  System.out.println(a + " " + b);
}

public static void swap(Account c, Account d)
{
      Account tmp;
      tmp = c;
      c = d;
      c.setBalance(0);
      d = tmp;
}
```

ted,123,100 mary,456,99
ted,123,100 mary,0,99

Output

Example 2: Swapping 2 Account

```
public static void main(String[] arg)
{
  Account a = new Account("ted", 123, 100);
  Account b = new Account("mary", 456, 99);

  System.out.println(a + " " + b);
  swap(a, b);
  System.out.println(a + " " + b);
}

public static void swap(Account c, Account d)
{
    Account tmp;
    tmp = (Account) c.clone();
    c.changeTo(d);
    d.changeTo(tmp);
}
```

Driver

```
public class Account {
   …
 public Object clone() {
    Account copy = new Account();
    copy.acctNumber = this.acctNumber;
    copy.balance = this.balance;
    copy.name=(this.name).substring(0);
    return copy;
  }
 public void changeTo(Account b) {
    this.acctNumber = b.acctNumber;
    this.balance = b.balance;
    this.name = (b.name).substring(0);
  }
```

```
ted,123,100 mary,456,99
mary,456,99 ted,123,100
```

Output

Example 2: Swapping 2 Account

```
public static void main(String[] arg)
{
  Account a = new Account("ted", 123, 100);
  Account b = new Account("mary", 456, 99);

  System.out.println(a + " " + b);
  swap(a, b);
  System.out.println(a + " " + b);
}

public static void swap(Account c, Account d)
{
    Account tmp;
    tmp = (Account) c.clone();
    c.changeTo(d);
    d.changeTo(tmp);
     c.setBalance(0);
}
```

Driver

```
public class Account {
   …
 public Object clone() {
    Account copy = new Account();
    copy.acctNumber = this.acctNumber;
    copy.balance = this.balance;
    copy.name=(this.name).substring(0);
    return copy;
  }
 public void changeTo(Account b) {
    this.acctNumber = b.acctNumber;
    this.balance = b.balance;
    this.name = (b.name).substring(0);
  }
```

ted,123,100 mary,456,99
Mary,0,99 ted,123,100

Output

Example 3: Swapping 2 Account

```
public static void main(String[] arg)
{
  Account a = new Account("ted", 123, 100);
  Account b = new Account("mary", 456, 99);

  System.out.println(a + " " + b);
  swap(a, b);
  System.out.println(a + " " + b);
}

public static void swap(Account c, Account d)
{
    Account tmp;
    tmp = (Account) c.clone();
    c = (Account) d.clone();
    d = (Account) tmp.clone();
}
```

Driver

```
public class Account {
   …
  public Object clone() {
    Account copy = new Account();
    copy.acctNumber = this.acctNumber;
    copy.balance = this.balance;

copy.name=(this.name).substring(0);
    return copy;
  }
```

Account.java

```
ted,123,100 mary,456,99
ted,123,100 mary,456,99
```

Output

Example 3: Swapping 2 Account

```
public static void main(String[] arg)
{
  Account a = new Account("ted", 123, 100);
  Account b = new Account("mary", 456, 99);

  System.out.println(a + " " + b);
  swap(a, b);
  System.out.println(a + " " + b);
}

public static void swap(Account c, Account d)
{
    Account tmp;
    tmp = (Account) c.clone();
    c = (Account) d.clone();
    d = (Account) tmp.clone();
     c.setBalance(0);
}
```

Driver

```
public class Account {
   …
  public Object clone() {
    Account copy = new Account();
    copy.acctNumber = this.acctNumber;
    copy.balance = this.balance;

copy.name=(this.name).substring(0);
    return copy;
  }
```

Account.java

```
ted,123,100 mary,456,99
ted,123,100 mary,456,99
```

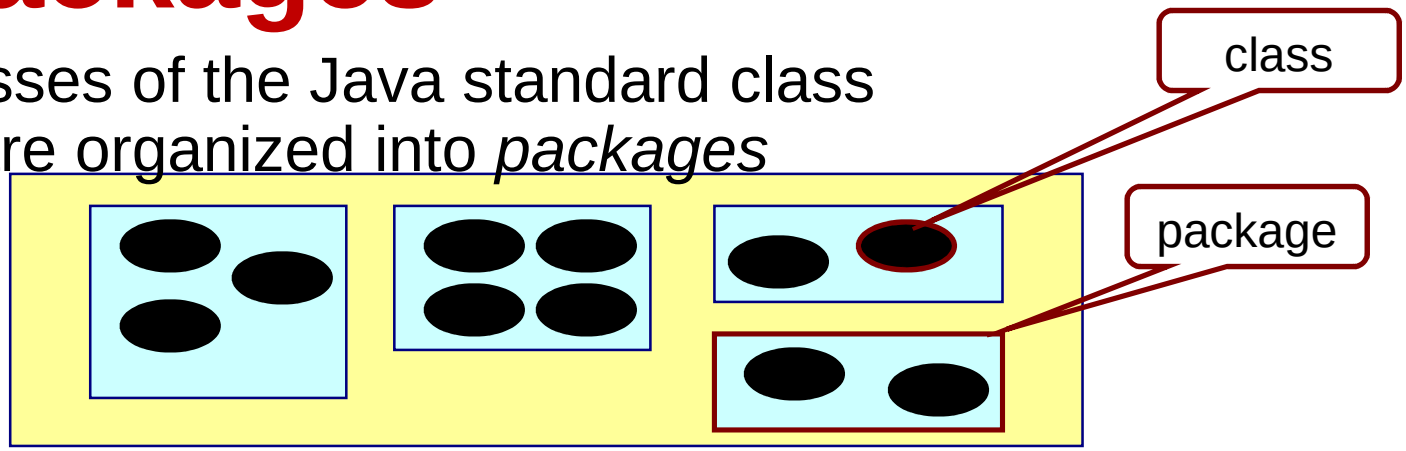
Output

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`**

5. Packages

The classes of the Java standard class library are organized into *packages*



Some standard packages :

Package	Purpose
java.lang	General support (manipulation of primitive types…) included by default in any Java program
java.applet	to create applets for the web
java.awt	Graphics and graphical user interfaces AWT = Abstract Windowing Toolkit
javax.swing	Additional graphics capabilities and components

The import Declaration

To use a class from a package

you can *import* the class from the package

```
import java.util.Random;
```

or import **all** classes from the package

```
import java.util.*;
```

Example: Java.lang.Math

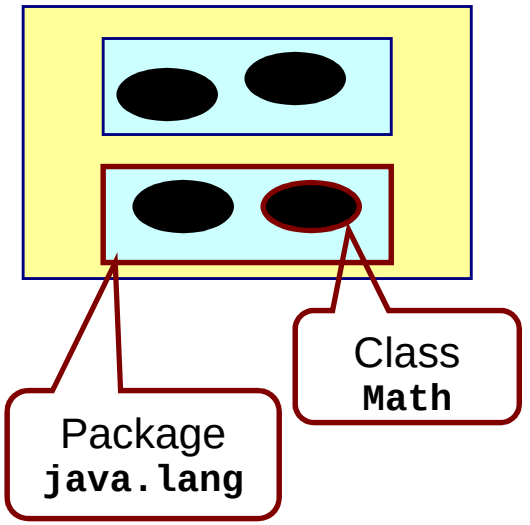
`java.lang.Math`

declares methods and math constants

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

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

```
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;
```

User-defined Packages

- To make a package:
  - group all the classes into a single directory
  - add `package package_name;` at the beginning of each file

  Some rules:

  Only blank lines and comments may precede the package statement

  Only the `.class` files must be in the directory

  If you have both `import` and `package` statements, put `package` before `import` statements

The program can be in a different directory from the package

Package Names and Directories

The name of a package = path name of the directory that contains the package classes

Java needs 2 things to find the directory for a package:

1. the name of the package
2. the value of the **CLASSPATH** variable

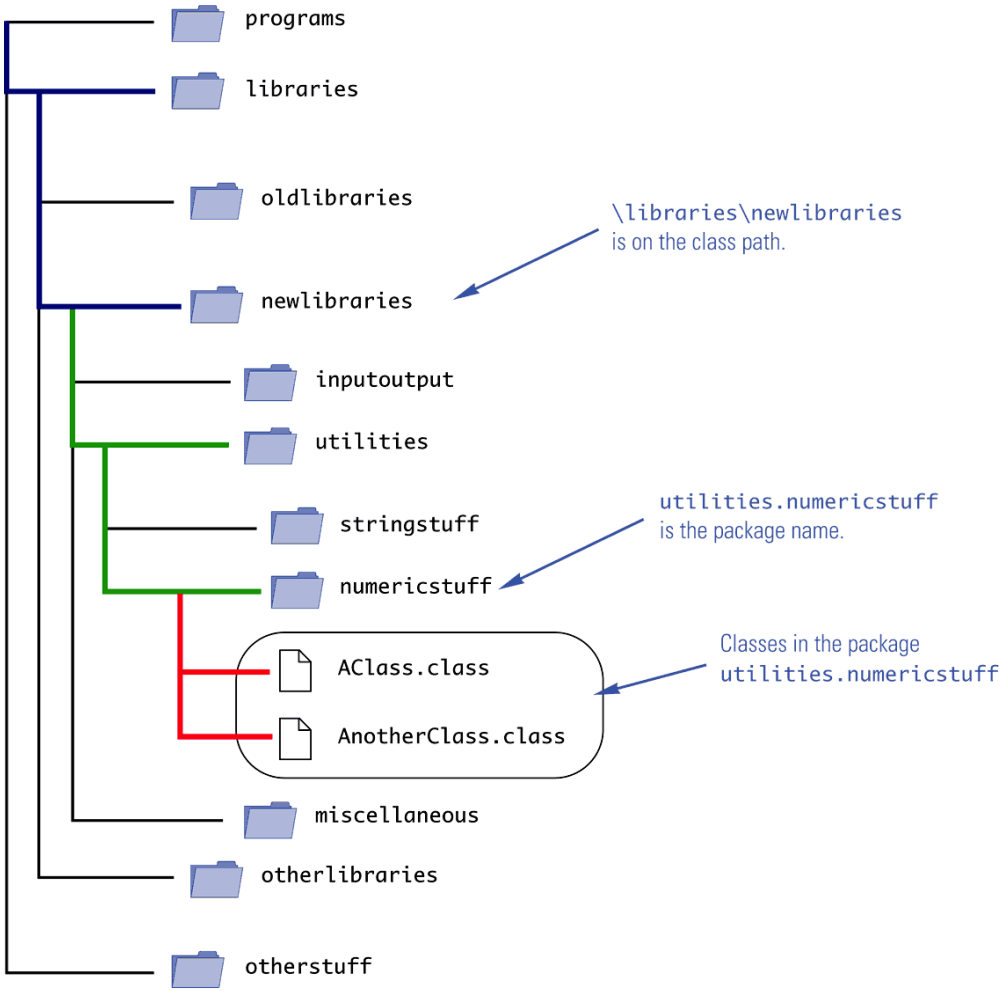
The **CLASSPATH** environment variable:

- is similar to the **PATH** variable from your OS
- is set in the same way for a given OS
- is set to the list of directories (including the current directory, "**.**") in which Java will look for packages

Java searches this list of directories in order, and uses the first directory on the list in which the package is found

A Package Name

Display 5.14 **A Package Name**



Subdirectories Are Not Automatically Imported

The import statement:

```
import utilities.numericstuff.*;
```

imports the `utilities.numericstuff` package only

The import statements:

```
import utilities.numericstuff.*;
import utilities.numericstuff.statistical.*;
```

import both the `utilities.numericstuff` and `utilities.numericstuff.statistical` packages

The Default Package

All the classes in the current directory belong to the *default package*

If the current directory (`.`) is in the **CLASSPATH**

all classes in the default package are available

Name Clashes

- Advantages of packages:
  keep class libraries organized
  provide a way to deal with *name clashes*, i.e., **when two classes have the same name**

- A name clash can be resolved:
  by using the *fully qualified name*
  ex: `package_name.ClassName`

- If the fully qualified name is used, it is no longer necessary to import the class

Javadoc

Comments in Java

```
// Can be single line comments
```

```
/* More than one line.
   Useful to "erase" a block
   of code from compilation */
```

```
/**
  * A Javadoc comment
  * To generate nice HTML documentation
 */
```

What is Javadoc?

A standard for documenting Java programs

`javadoc.exe`

- generates documentation of your Java classes in a standard format
- comes with the JDK.

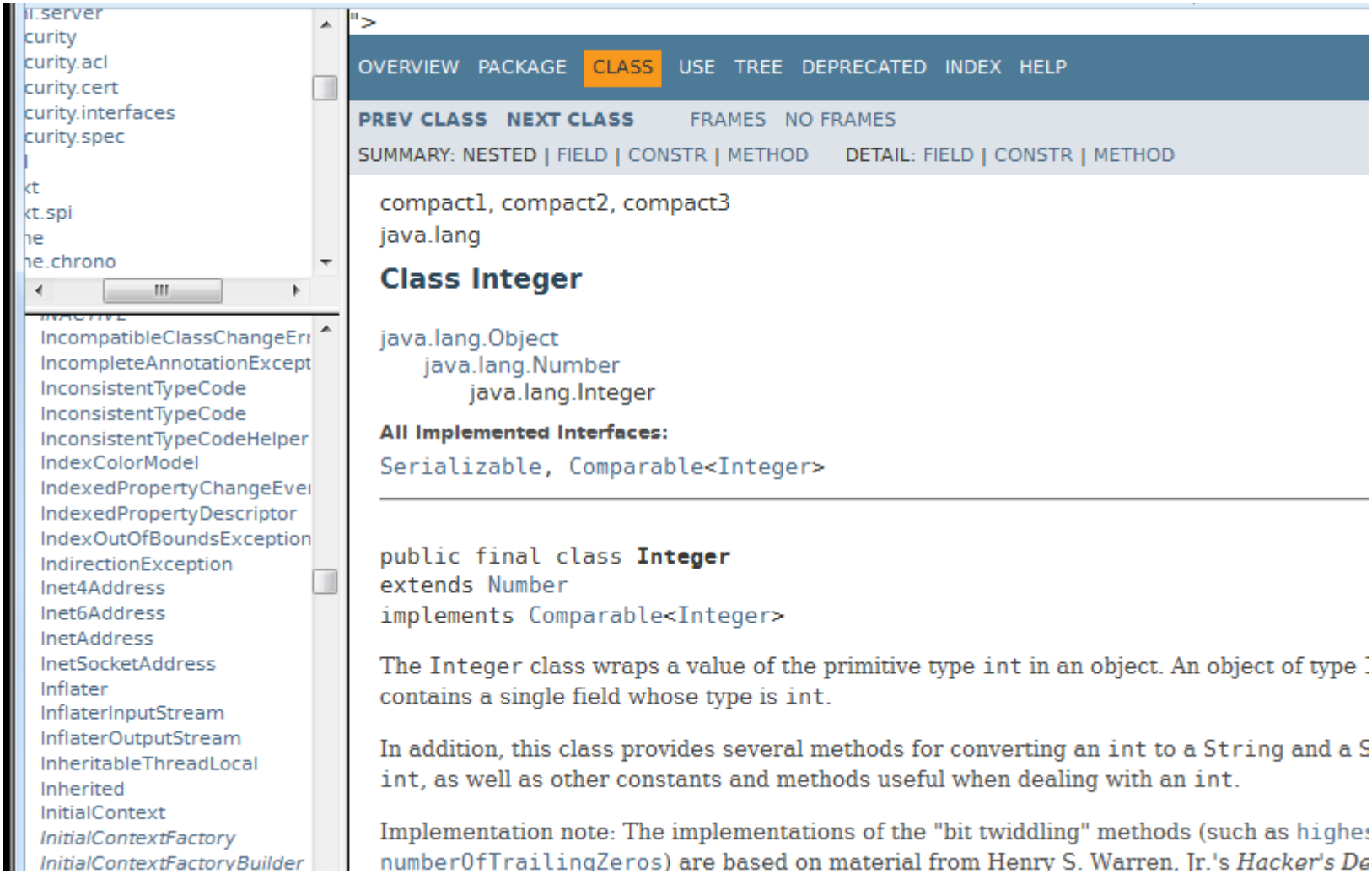
For each **`X.java`**, it will create an **`X.html`**

- plus a couple of additional pages (index, ..)

ex: If you wrote **`MyClass.java`**

- then running "**`javadoc MyClass.java`**"
- generates a file "**`MyClass.html`**"
- that contains properly formatted comments

- The output from Javadoc looks exactly like the API documentation… since that's the way it was generated!



General form of a Javadoc comment

```
/**
* One sentence ending with a period describing the purpose.
* Additional lines giving
* more details (html tags can be included)
*
* javadoc tags to specify more specific information,
* such as parameters and return values for a method
*
* @Tag Description ...
* @Tag Description ...
*
 */
```

Tags

- Allow you to specify specific information used in the HTML file
- Most common tags:

  For files, classes and interfaces:

  ```
  @author Name
  @version Version number
  ```

  For methods:

  ```
  @param name description
  @return description
  @exception exceptionClass description
  @deprecated description
  ```

  For everything:

  ```
  @see relatedReference (ex. other class name)
  ```

Documenting Files, Classes and Interfaces

- Javadoc comments go immediately before the class or interface (or top of file)

ex:

```
/**
  * A coin value and its name.
  * @author YourNsmr
  * @version 1.1
  * @see JavadocForger
  */
public class JavadocCoin {
…
}
```

```
javadoc -author -version Javadoc.java
```

Documenting Methods

Javadoc comments go immediately before the method definition

Common tags :

@param <name of parameter> <description>

@return <description>

ex:

```
/**
 * This method does this.
 * @param coinValue Initial coin value
 * @param coinName Initial coin name
 * @return The value of the coin
 */
  public double someMethod(double coinValue, String coinName){
   …
  }
```

Javadoc

Advantages:

- looks very nice…
- provides a consistent look and feel to API documents
- when source code is changed:
  - the Javadoc comments can be changed in the source, at the same time.
  - the external documentation is then easily re-generated.