Java™ Education & Technology Services

Java Programming

Course Outline

- Lesson 1: Introduction to Java
- Lesson 2: Basic Java Concepts
- Lesson 3: Applets
- Lesson 4: Data Types & Operators
- Lesson 5: using Arrays & Strings
- Lesson 6: Controlling Program Flow

Course Outline

- Lesson7: Java Exception
- Lesson 8: Interfaces
- Lesson 9: Multi-Threading
- Lesson 10: Inner class
- Lesson 11: Event Handling

Lesson 1

Introduction To Java

Brief History of Java

- Java was developed by Sun Microsystems in **may 1995.**
- The Idea was to create a language for controlling any hardware, but it was too advanced.
- A team - **that was called the Green Team** - was assembled and lead by **James Gosling**.
- Platform and OS **Independent** Language.
- **Free** License; cost of development is brought to a minimum.

Brief History of Java

- From mobile phones to handheld devices, games and navigation systems to e-business solutions, **Java is everywhere!**

- Java can be used to create:
  - Desktop Applications,
  - Web Applications,
  - Enterprise Applications,
  - Mobile Applications,
  - Smart Card Applications.
  - Embedded Applications (Rasburry PI)
  - Java SE Embedded

Java Principles

- Primary goals in the design of the Java programming language:

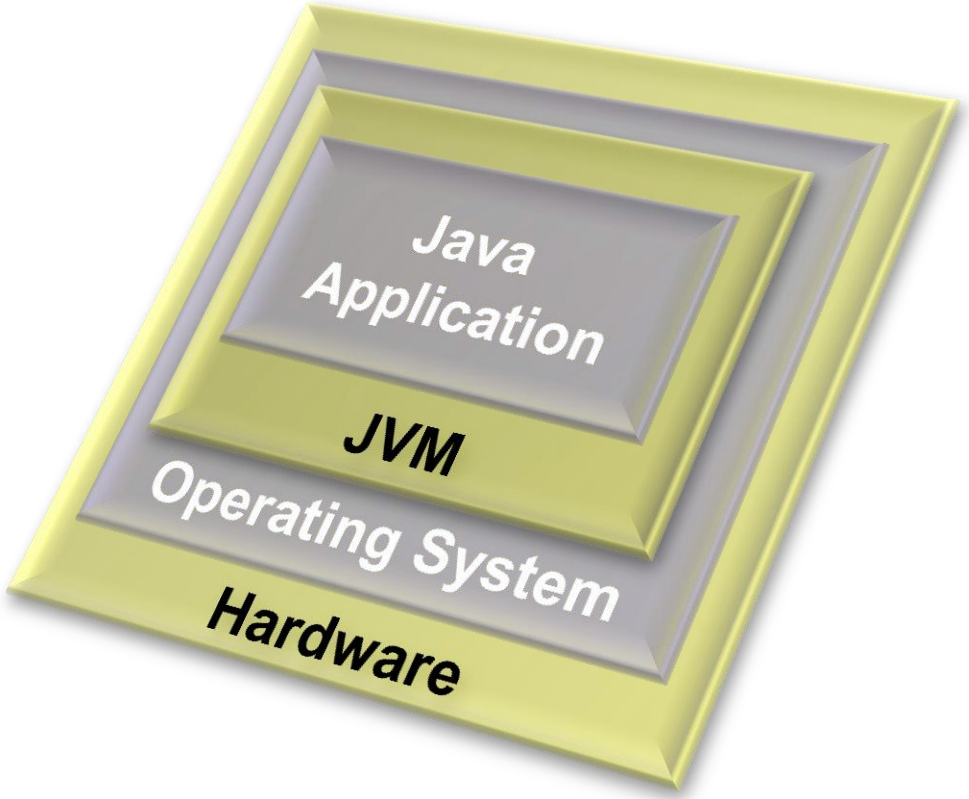
- Simple
- Object oriented
- Distributed
- Multithreaded
- Dynamic
- Portable
- High performance
- Robust
- Secure

Java Features

- Java is easy to learn!
  - Syntax of C++
  - Dynamic Memory Management (Garbage Collection)
  - No pointers

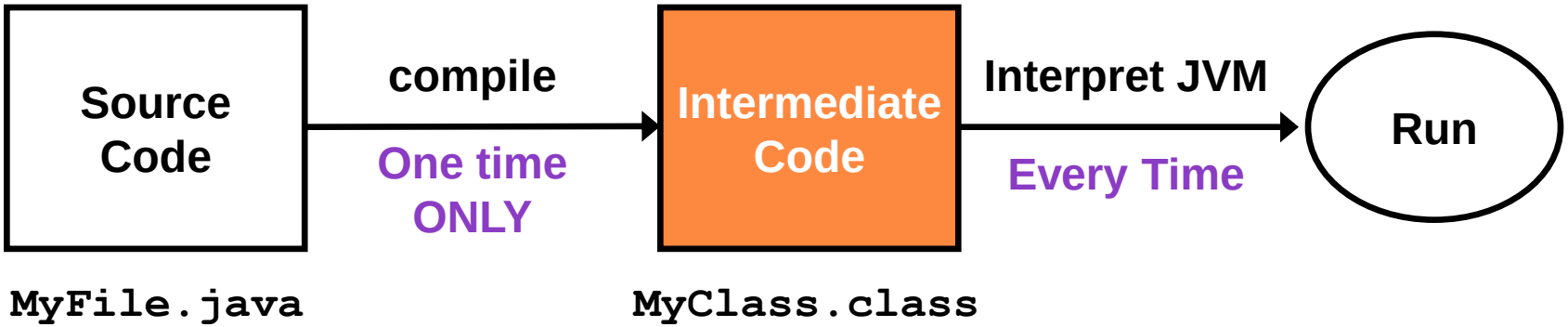
Java Features cont'd

- Machine and Platform Independent (Architecture Neutral)



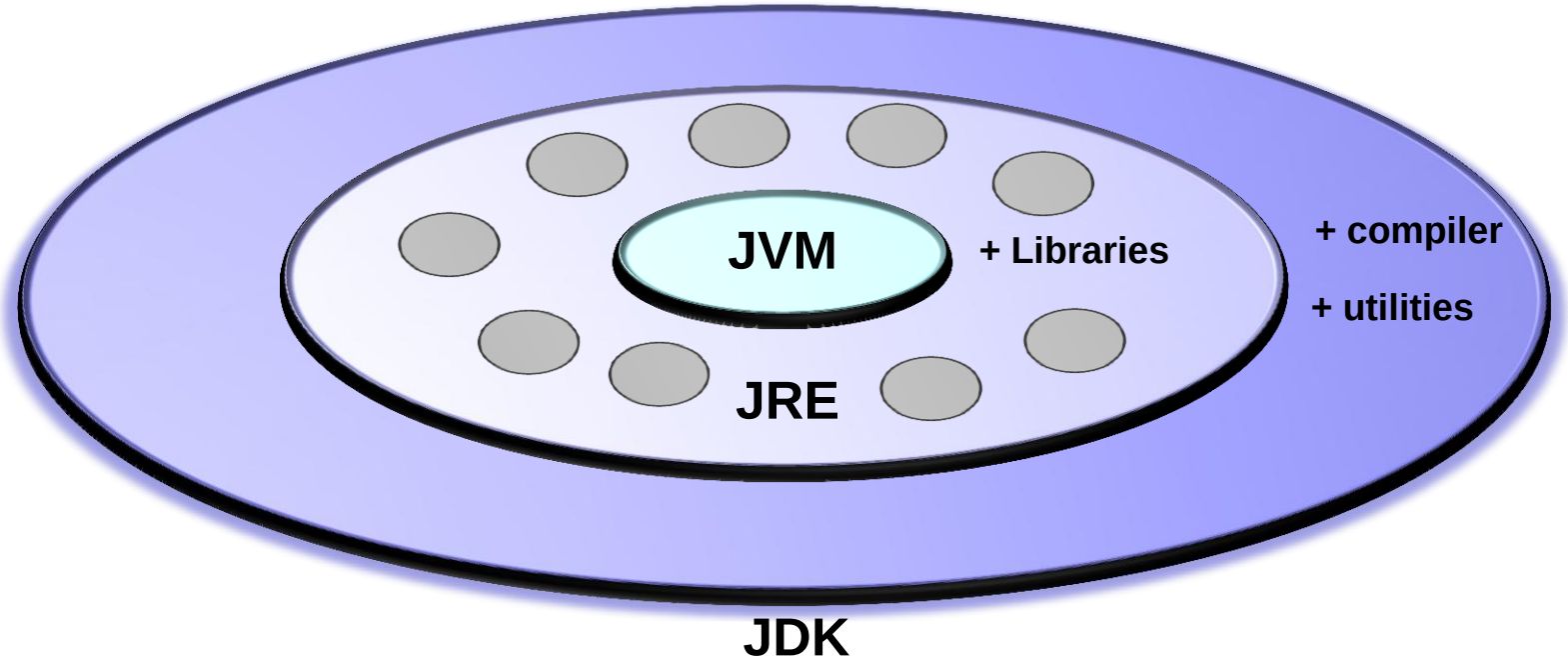
Java Features cont'd

- Java is both, compiled and interpreted



Java Features cont'd

- Java depends on dynamic linking of libraries



Java Features cont'd

- Java is fully Object Oriented
  - Made up of Classes.
  - No multiple Inheritance.
- Java is a multithreaded language
  - You can create programs that run multiple threads of execution in parallel.
    - Ex: GUI thread, Event Handling thread, GC thread
- Java is networked
  - Predefined classes are available to simplify network programming through Sockets(TCP-UDP)

Installing JDK

- Download the JDK:
  - If you use Solaris, Linux, Windows, or Mac point your browser to http://www.oracle.com/technetwork/java/javase/downloads/index.html to download the JDK.
  - Look for version 7.0 or later, and pick your platform.
  - After downloading the JDK, follow the platform-dependent installation directions.

    http://docs.oracle.com/javase/7/docs/webnotes/install/index.html

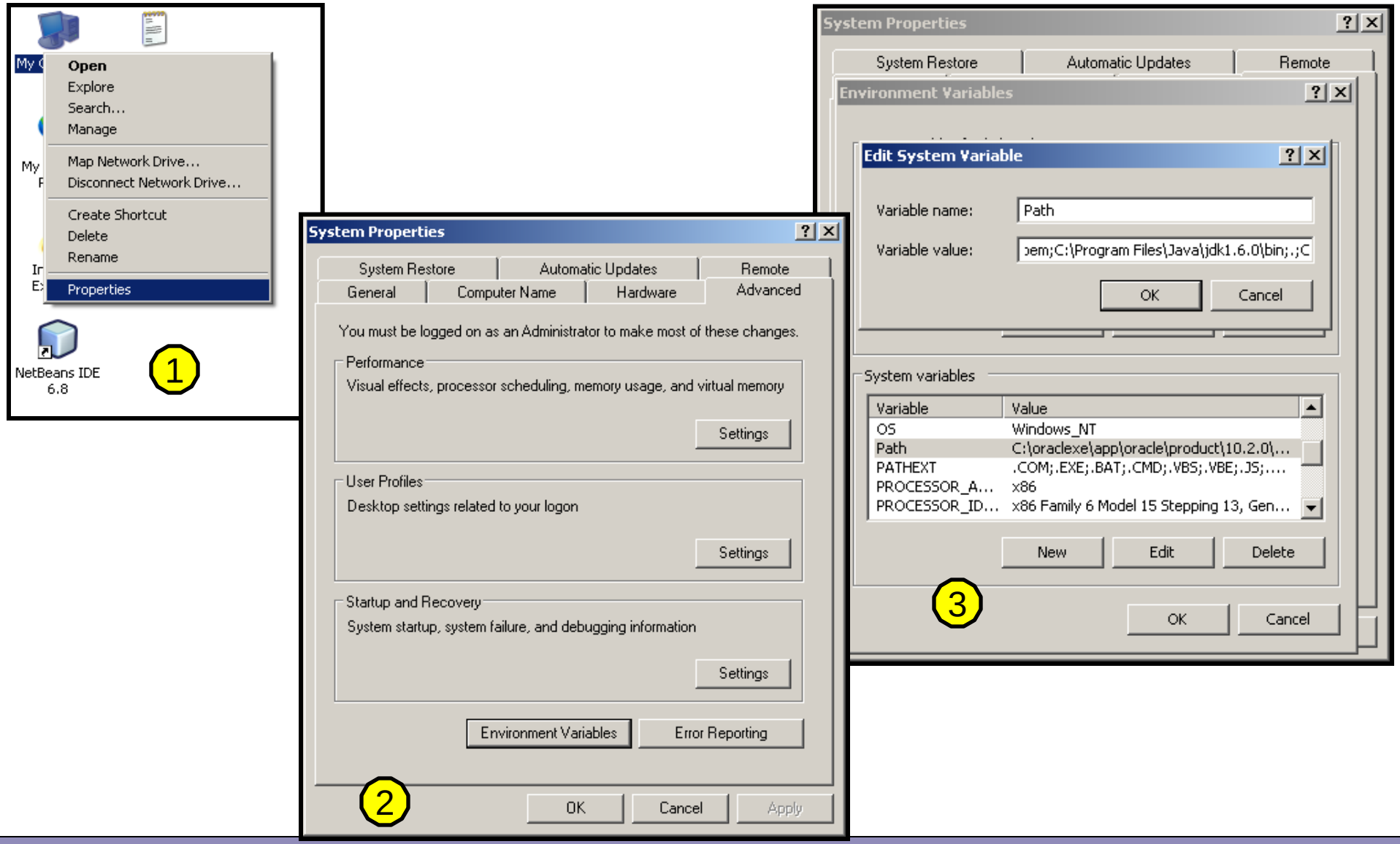
Java Environment Setup "windows"

- Once you installed Java on your machine,
  - you would need to set environment variables to point to correct installation directories:
    - Assuming you have installed Java in

      c:\Program Files\java\jdk directory\bin

    - Right-click on **'My Computer'** and select **'Properties'**.
    - Click on the **'Environment variables'** button under the **'Advanced'** tab.
    - Now alter the **'Path'** variable so that it also contains the path to the Java executable.

Java Environment Setup "windows"



Lesson 2

Basic Java Concepts

How to create a class?

- To define a class, we write:

```
<access-modifier>* class <name>
{
        <attributeDeclaration>*
        <constructorDeclaration>*
        <methodDeclaration>*
}
```

- Example:

```
class StudentRecord {
        //we'll add more code here later
}
```

Coding Guidelines

- Think of an appropriate name for your class.
  - Don't use XYZ or any random names.
- Class names starts with a CAPITAL letter.
  - not a requirement it is a convention

Declaring Properties (Attributes)

- declare a certain attribute for our class, we write,

```
<access-modifier>* <type> <name> [= <default_value>];
```

- Example:

```
class StudentRecord {
                // Instance variables
                public String       name;
                public String       address;
                private int         age    =      15;
                /*we'll add more code here later */
}
```

Declaring Properties (Attributes)

- **Access modifiers:**

1. **Public attributes:**
   - The access availability inside or outside the class.
2. **Private attributes:**
   - The access availability within the class only.

Declaring Methods

- declare a certain method for our class, we write,

```
<modifier>* <Return type> <name> ([<Param Type> <Param
Name>]*)
{
        <Statement>*
}
```

- Example:

```
class StudentRecord {
        private String name;
        public String getName(){ return name; }
        public void setName(String str){ name=str; }
        public static String getSchool(){.....}
}
```

Declaring Methods

- The following are characteristics of methods:
  - It can return one or no values
  - It may accept as many parameters it needs or no parameter at all.
    - Parameters are also called arguments.
  - After the method has finished execution, it goes back to the method that called it.
  - Method names should start with a small letter.
  - Method names should be verbs.

Declaring Properties (Methods)

- **Access modifiers:**

1. **Public method:**
   - The access availability inside or outside the class.
2. **Private method:**
   - The access availability within the class only.
3. **Static method:**
   - Methods that can be invoked without instantiating a class.
   - To call a static method, just type,

```
Classname.staticMethodName(params);
```

Big Example

```
class Student{
   String firstName,lastName;
   int age;
   double mathScore;
   double scienceScore;
   int getAge(){ return age; }
   void setAge(int g){ age=g; }
   public static String getSchool(){//return school
   name}
   public double average(){
        double avg=0;
        avg=(mathScore+scienceScore)/2;
        return avg;
   }
}
```

Create Object Instance

- To create an object instance of a class,
  - we use the **`new`** operator.
- For example,
  - if you want to create an instance of the class Student, we write the following code,

```
Student s1 = new Student();
```

- The new operator
  - Allocates a memory for that object and returns a reference of that memory location to you.
  - When you create an object, you actually invoke the class' constructor.

Accessing members of class

- To access members of class:

```
class Test {
        void testMethod(){
                Student s1 = new Student();
                s1.setAge(10);
                double d;
                d = s1.average();
                String s = Student.getSchool();
        }
}
```

First Java Application

```
class HelloWorld
{
  public static void main(String[] args)
  {
      System.out.println("Hello Java");
  }
}
```

File name: `hello.java`

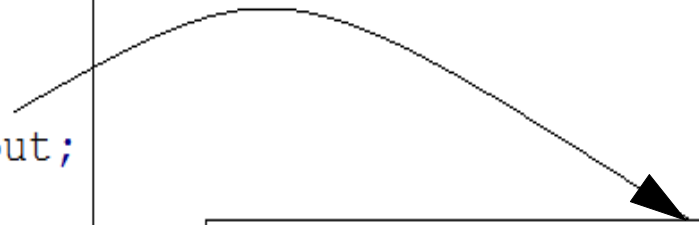
First Java Application cont'd

- The **`main()`** method:
  - Must return void.
  - Must be static.
    - because it is the first method that is called by the Interpreter (**`HelloWorld.main(..)`**) even before any object is created.
  - Must be public to be directly accessible.
  - It accepts an array of strings as parameter.
    - This is useful when the operating system passes any command arguments from the prompt to the application.

System.out.println("Hello");

- **out** is a static reference that has been created in class **System**.
- **out** refers to an object of class **PrintStream**. It is a ready-made stream that is attached to the standard output (i.e. the screen).

```
public class System
{
   public static PrintStream out;
   ..............
}
```



```
public class PrintStream
{
   public void print(String str)
   {...........}

   public void println(String str)
   {...........}
}
```

Standard Naming Convention
"The Hungarian Notation."

- Class names:

  `MyTestClass` , `RentalItem`

- Method names:

  `myExampleMethod()` , `getCustomerName()`

- Variables:

  `mySampleVariable` , `customerName`

- Constants:

  `MY_STATIC_VAR` , `MAX_NUMBER`

- Package:

  `pkg1` , `util` , `accesslayer`

Compiling and Running a Java Application

- **To compile:**

```
Prompt> javac hello.java
```

- If there are no compiler errors, then the file `HelloWorld.class` will be generated.
- **To run:**

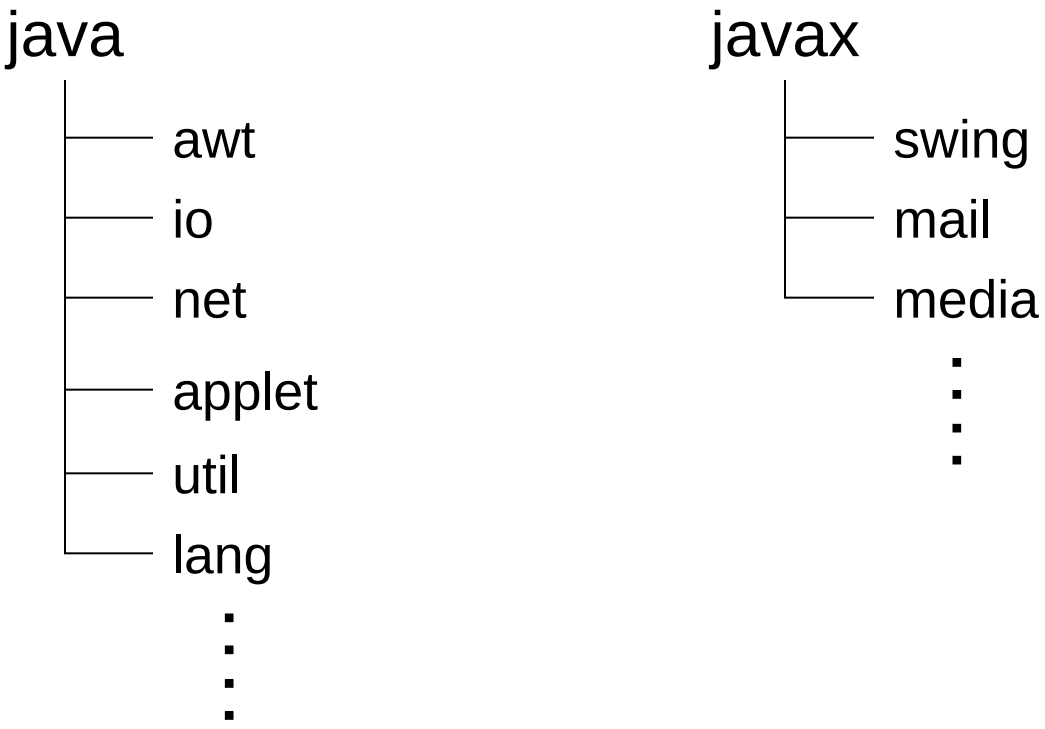
```
Prompt> java HelloWorld
Hello Java
Prompt>
```

Java Structure

- Classes are placed in packages.
- We must import any classes that we will use inside our application.
- Classes that exist in package `java.lang` are imported by default.
- Any Class by default extends `Object` class.

Java Structure cont'd

- The following are some package names that contain commonly used classes of the Java library:



Specifying a Package

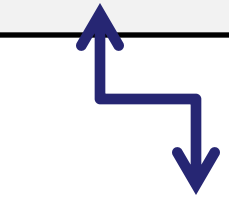
- If no package is specified,
  - then the compiler places the .class file in the default package (i.e. the same folder of the .java file).
- To specify a package for your application,
  - write the following line of code at the beginning of your class:

```
package mypkg;
```

Specifying a Package

- To compile and place the .class in its proper location:

```
Prompt> javac -d . hello.java
```



Current Directory

- To run:

```
Prompt> java mypkg.HelloWorld
```

JAR File

- Packages can be brought together in one compressed JAR file.
- The classes of Java Runtime Libraries (JRE) exist in **`rt.jar`**.
- JAR files can be made executable by writing a certain property inside the **manifest.mf file** that points to the class that holds the **`main(..)`** method.

How to make JAR file

- To create a compressed JAR file:

```
prompt> jar cf <archive_name.jar> <files>
```

- **Example:**

```
prompt> jar cf App.jar HelloWorld.class
```

How to make JAR file cont'd

- To create an executable JAR file:
  1. Create text file that list the main class.

     "The class that has the main method"
  2. Write inside the text file this text:

     Main-Class: <class name>
  3. Then run the jar utility with this command line:

```
prompt>jar cmf <text-file> <archive_name.jar>
   <files>
```

Lab Assignments

1. Simple Prompt Application

- Create a simple non-GUI Application that prints out the following text on the command prompt:

  Hello Java

- **Note:** specify package and create executable jar file.
- **Bonus:** Modify the program to print a string that is passed as an argument from the command prompt.

2. Simple Prompt Application

- Create a simple non-GUI Application that represent complex number and has two methods to add and subtract complex numbers:

Complex number: **x + yi** , **5+6i**

Lesson 3

Applet

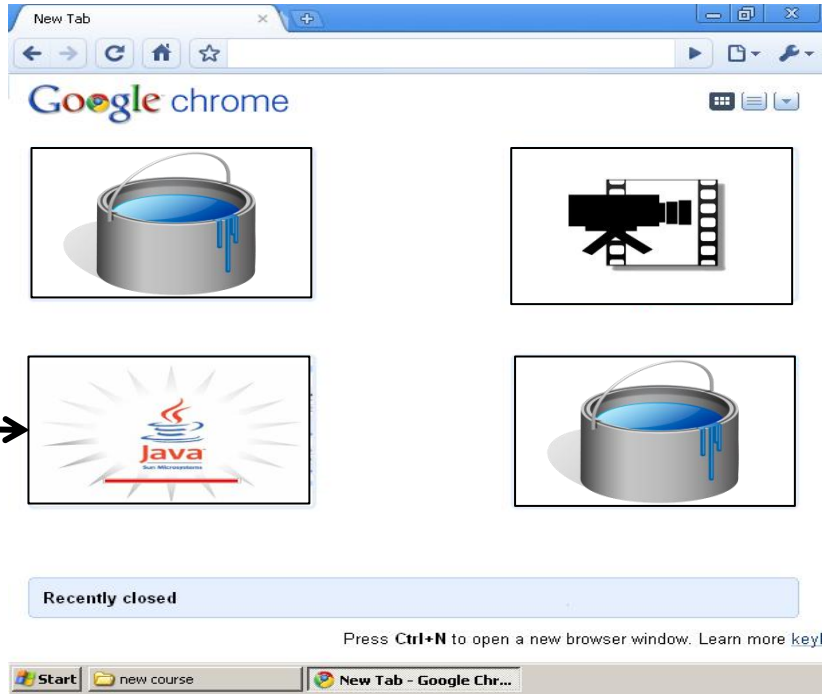
Overview

Web Server

(www.abc.com)

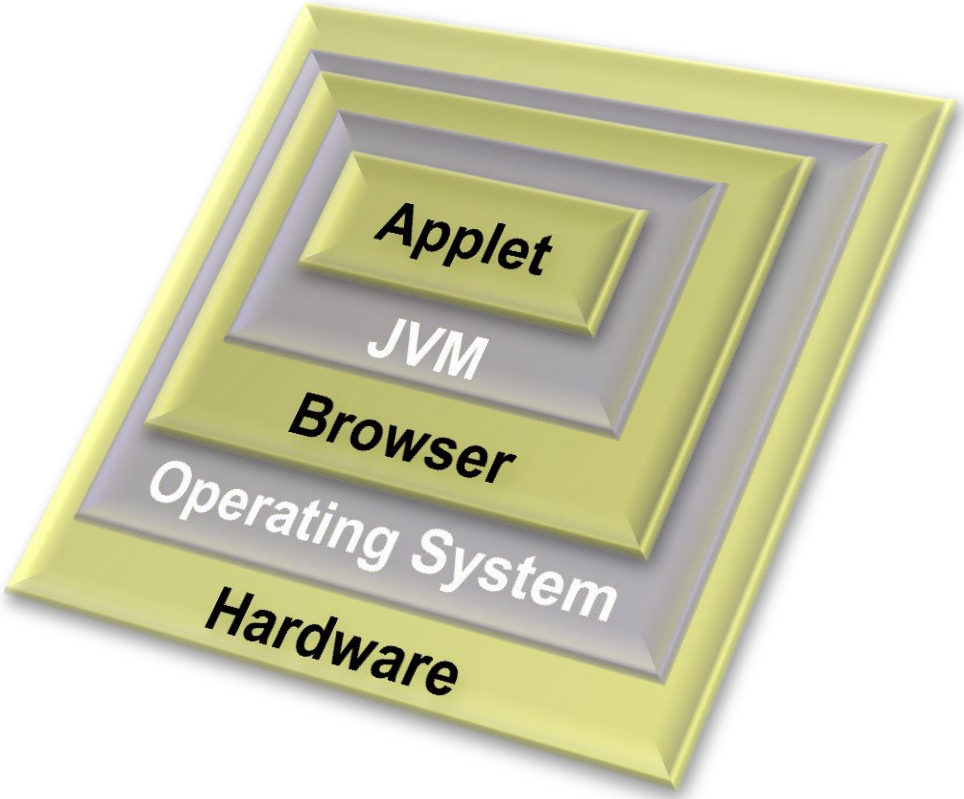
Download MyApplet.class

www.abc.com\index.html



Applet Features

- Machine and Platform Independent



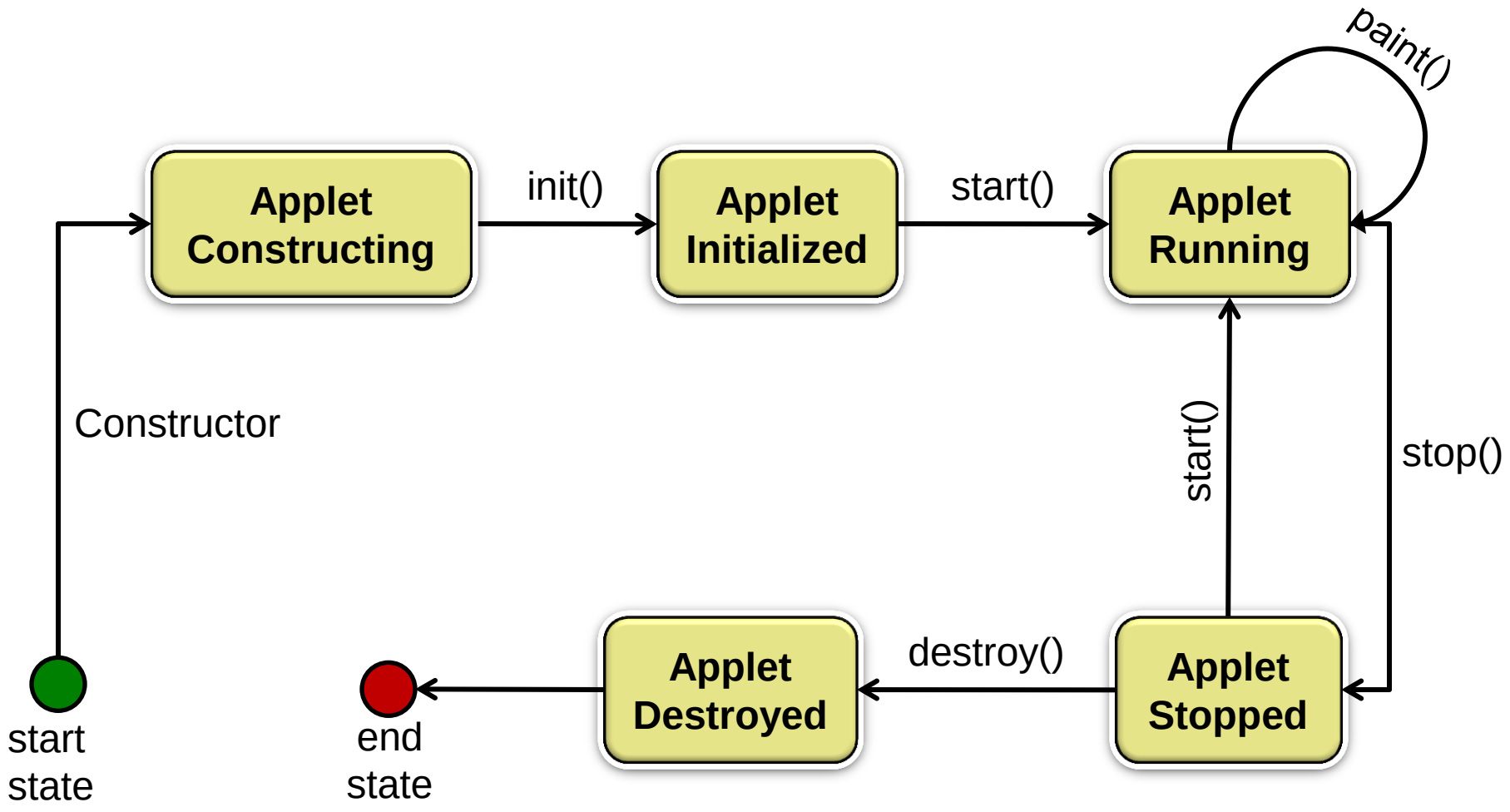
Applets

- An Applet is a client side Java program that runs inside the web browser.
- The .class file of the applet is downloaded from the web server to the client's machine
- The JVM interprets and runs the applet inside the browser.

Applet Security

- In order to protect the client from malformed files or malicious code, the JVM enforce some security restrictions on the applet:
  - Syntax is checked before running.
  - No I/O operations on the hard disk.
  - Communicates only with the server from which it was downloaded.
- Applets can prompt the client for additional security privileges if needed.

Applet Life Cycle



Applet Life Cycle

- **The life cycle of Applet:**

- **`init()`:**
  - called when the applet is being initialized for the first time.

- **`start()`:**
  - called whenever the browser's window is activated.

- **`paint(Graphics g)`:**
  - called after **`start()`** to paint the applet, or
  - whenever the applet is repainted.

- **`stop()`:**
  - called whenever the browser's window is deactivated.

- **`destroy()`:**
  - called when the browser's window is closed.

Applet Life Cycle cont'd

- You can refresh the applet anytime by calling: **`repaint()`**,
  - which will invoke **`update(Graphics g)`** to clear the applet,
  - which in turn invokes **`paint(Graphics g)`** to draw the applet again.
- To create your own applet, you write a class that extends class **`Applet`**,
  - then you override the appropriate methods of the life cycle.

Basic Java Applet

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloApplet extends Applet{
  public void paint(Graphics g){
     g.drawString("Hello Java", 50, 100);
  }
}
```

Note: Your class must be made public or else the browser will not be able to access the class and create an object of it.

Basic Java Applet cont'd

- In order to run the applet we have to create a simple HTML web page, then we invoke the applet using the <applet> tag.
- The **<applet>** tag requires 3 mandatory attributes:
  - code
  - width
  - height
- An optional attribute is codebase, which specifies the path of the applet's package.

Basic Java Applet cont'd

- Write the following in an HTML file e.g. mypage.html:

```
<html>
  <body>
  <applet code="HelloApplet"
          width="400"  height="350"/>
  </body>
</html>
```

Compiling and Running an Applet

- Save the Hello Applet Program in your assignments folder in a file named: **HelloApplet.java**
  - When a class is made public, then you have to name the file after it.
- To compile write in cmd this command:

```
javac HelloApplet.java
```

- An applet is not run like an application.
- Instead, you browse the HTML file from your web browser, or by using the applet viewer:

```
appletviewer mypage.html
```

from the command prompt.

Lab Exercise

1. Basic Applet

- Create an applet that displays: **Hello Java.**

- **Bonus:** Try to pass some parameters from the HTML page to the applet. For example, display the parameters on the applet.

  Hint:

  use the self closing tag: `<param name= value= />`