Java™ Education & Technology Services

Java Programming

Course Outline

- Lesson 1: Introduction to Java
- Lesson 2: Basic Java Concept
- Lessons 3: Applets
- Lesson 4: Data Types & Operators
- Lesson 5: using Arrays & Strings
- Lesson 6: Controlling Program Flow
- Lesson7: Modifiers-Access Specifiers
  Essential Java Classes-
  Exception Handling

Presentation Outline

- 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 created 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 (Sun SPOT)

Java Principles

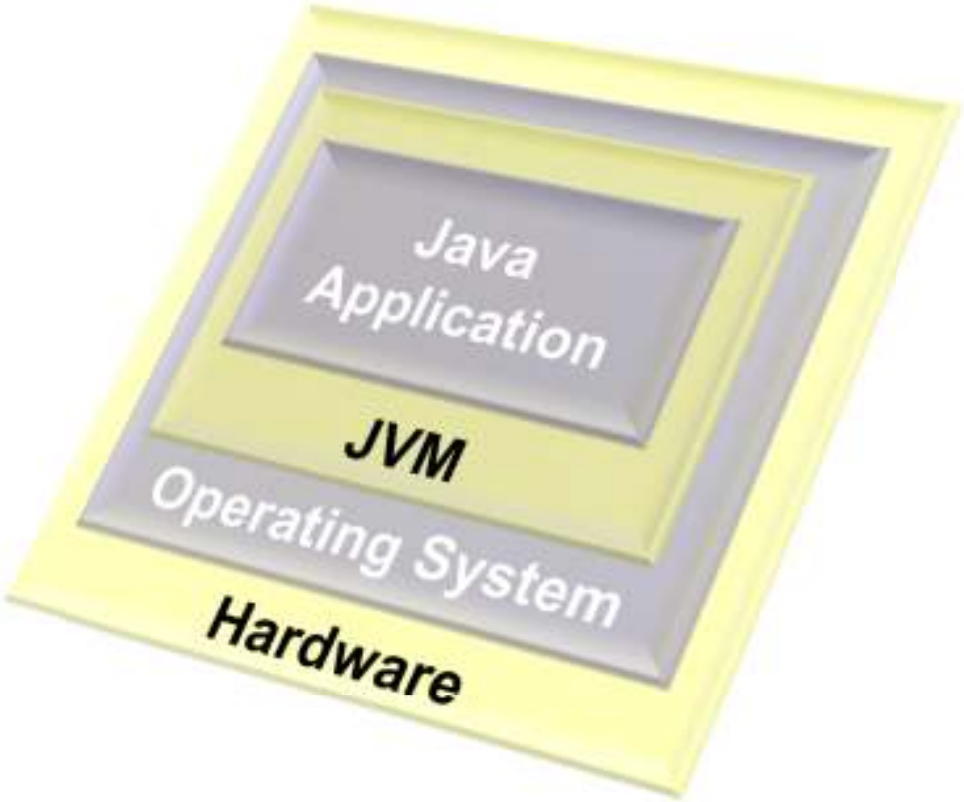
- Primary goals in the design of the Java programming language:
  - **Simple, object oriented, and easy to learn**.
  - **Robust and Secure**.
  - **Architecture neutral and portable**.
  - **Compiled and Interpreted.**
  - **Multithreaded**.
  - **Networked.**

Java Features

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

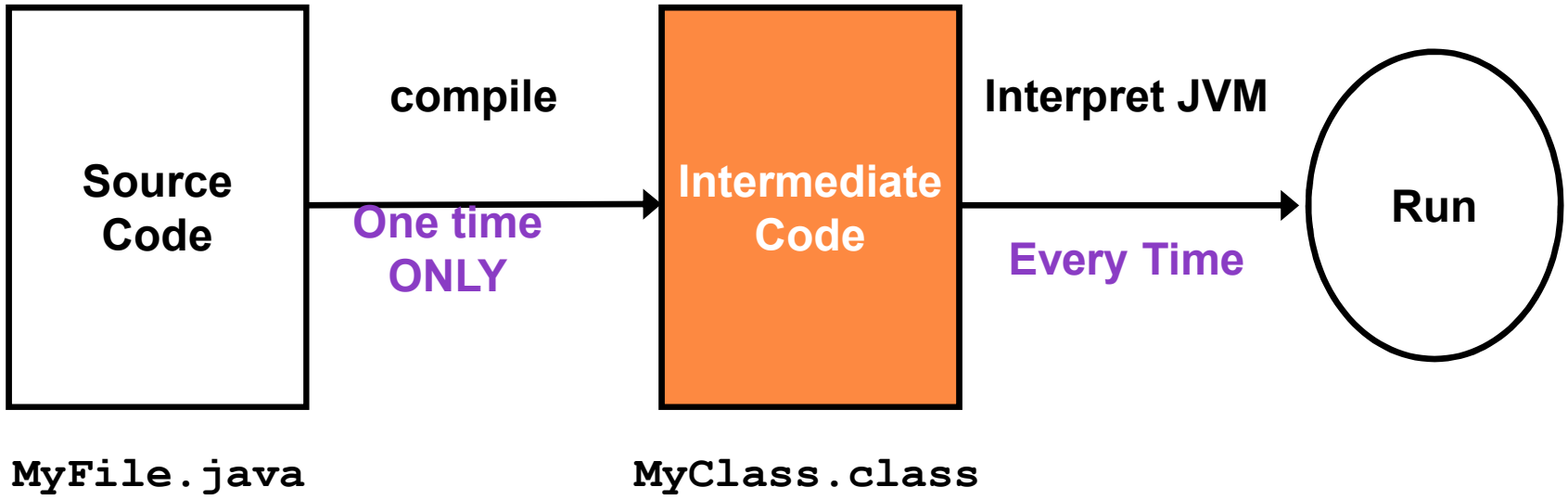
Java Features cont'd

- Machine and Platform Independent



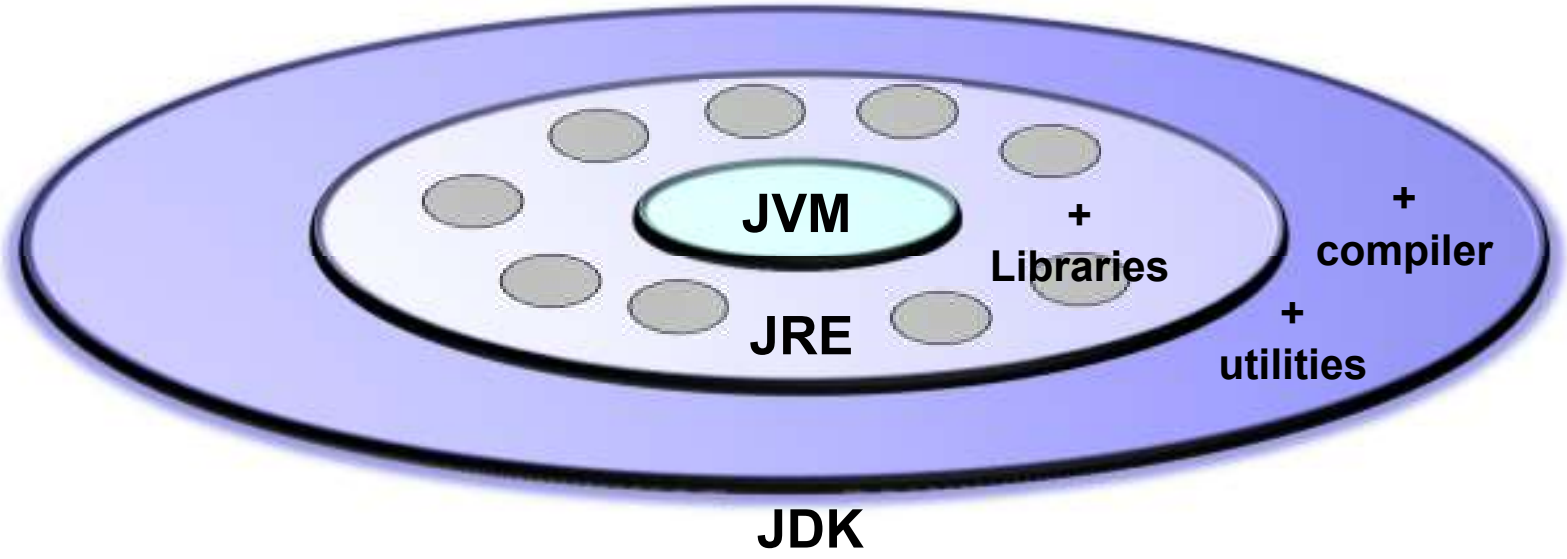
Java Features cont'd

- Java is both, compiled and interpreted



Java Features cont'd

- Java depends on dynamic linking of libraries



Java development Kit (JDK)

Java Features cont'd

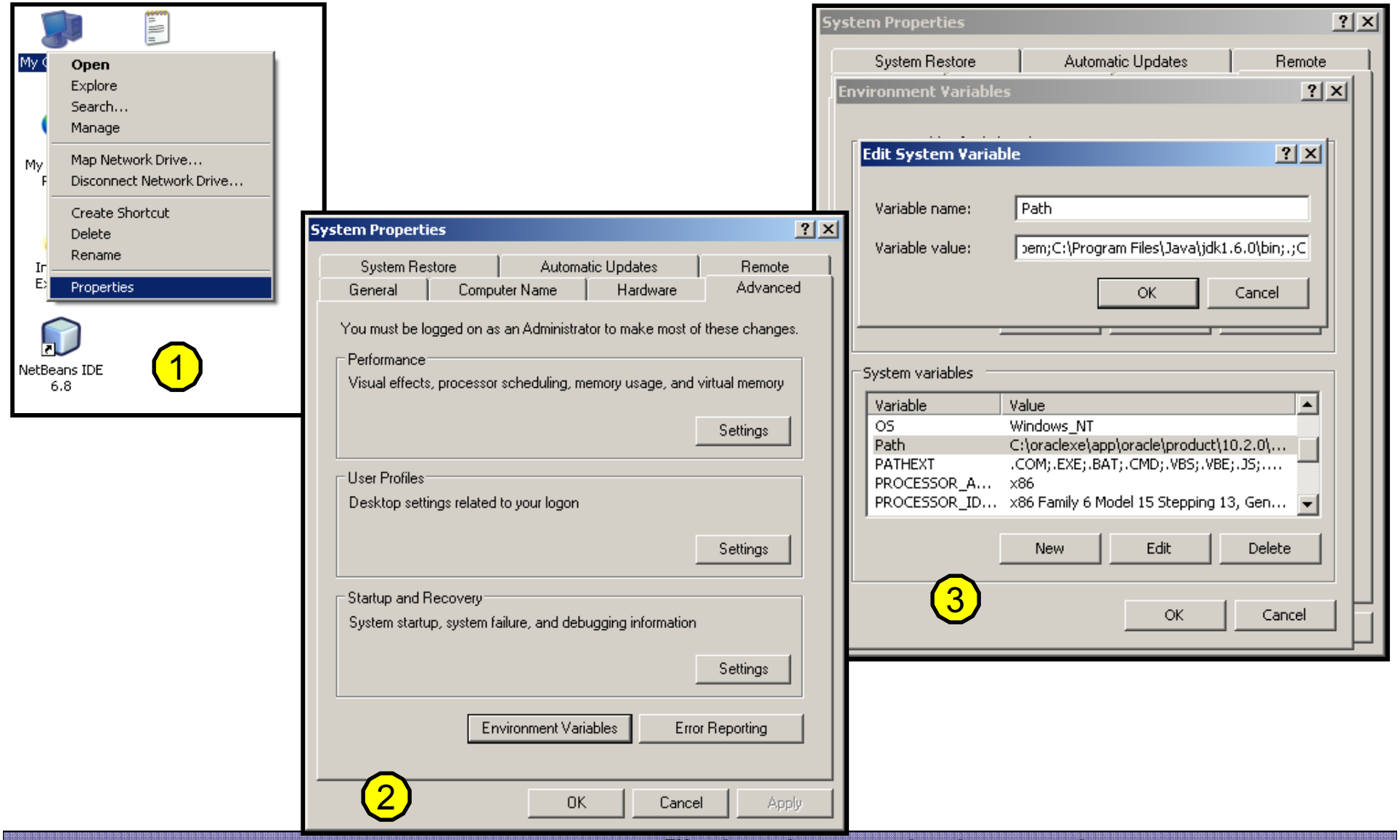
- Java is fully Object Oriented
  - Made up of Classes.
  - No multiple Inheritance.
- Java is a multithreaded language
  - You can create programs that run multiple threads of execution in parallel.
    - Ex: GUI thread, Event Handling thread, GC thread
- Java is networked
  - Predefined classes are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

Java Environment Setup

- 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



Lesson 2

Basic Java Concepts

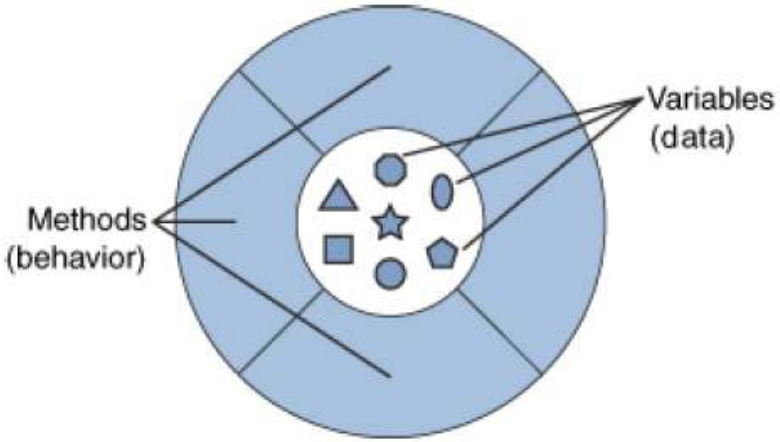
Introduction to OOP

- **What is OOP?**
  - OOP is mapping the real world to Software
  - OOP is a ***community*** of interacting agents called ***objects***.
  - Each object has a role to play.
  - Each object provides a ***service*** or performs an action that is used by other objects of the community.
  - Action is initiated by the transmission of a ***message*** to an object responsible for the actions.

- **What is OOP?**
  - All objects are instances of a ***class***.
  - The method invoked by an object is determined by the class of the receiver.
  - All objects of a given class use the same method in response to similar messages.

Introduction to OOP - Object

- **What is an Object?**
  - An object is a software bundle of variables and related methods.
- Object consist of:
  - **Data** (object's Attributes)
  - **Behavior** (object's methods)



Introduction to OOP - Class

- **What is a Class?**
  - A class is a blueprint of objects.
  - A class is an object factory.
  - A class is the template to create the object.
  - A class is a user defined datatype
- **Object:**
  - An object is an instance of a class.
  - The property values of an object instance is different from the ones of other object instances of a same class
  - Object instances of the same class share the same behavior (methods).

Introduction to OOP – Object & Class

- ***Class*** reflects concepts.
- ***Object*** reflects instances that embody those concepts.

class

object

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.
  - 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}

  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");
  }
}
```

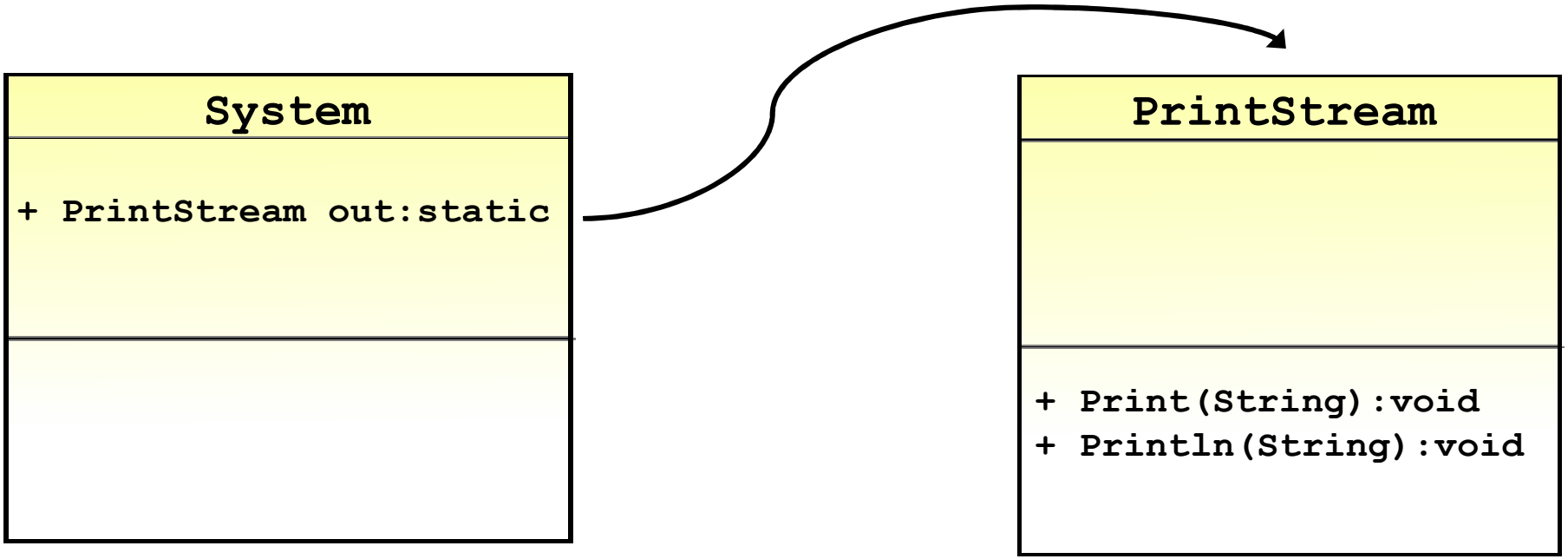
File name: `hello.java`

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).



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:**

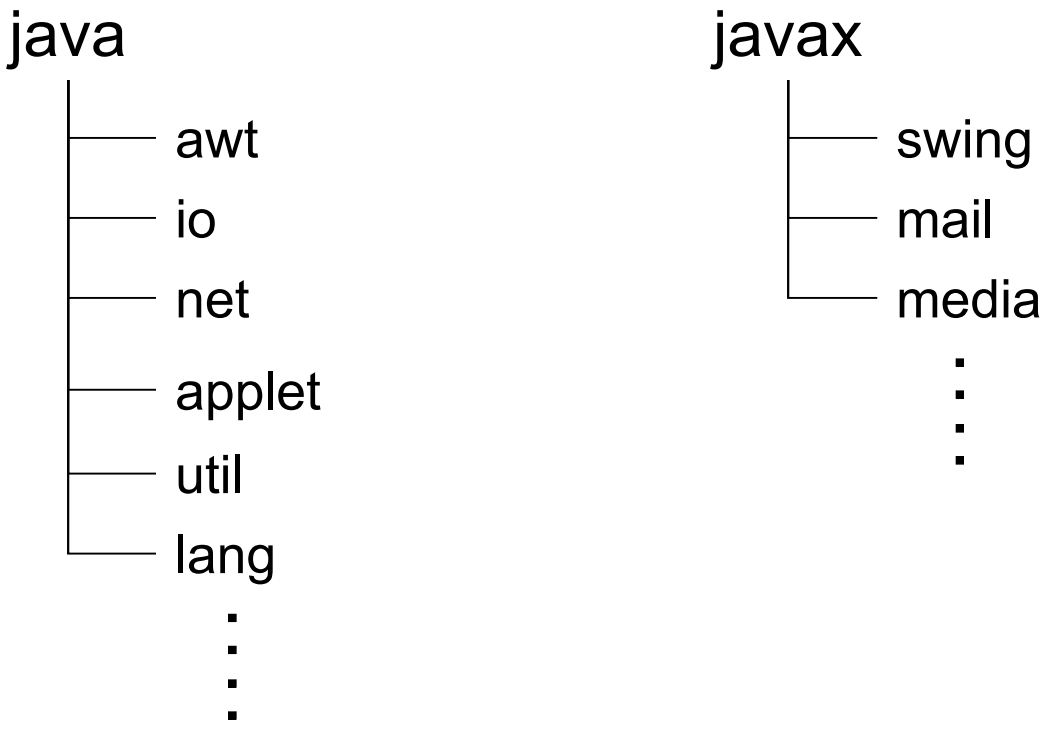
```
Prompt> java HelloWorld
Hello Java
Prompt>
```

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).

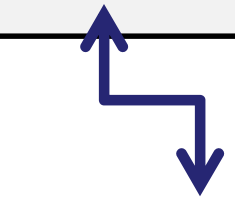
- To specify a package for your application,
  - write the following line of code at the beginning of your class:

```
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>
```

Or without manifest file:

```
prompt>jar cef <entry-point> <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

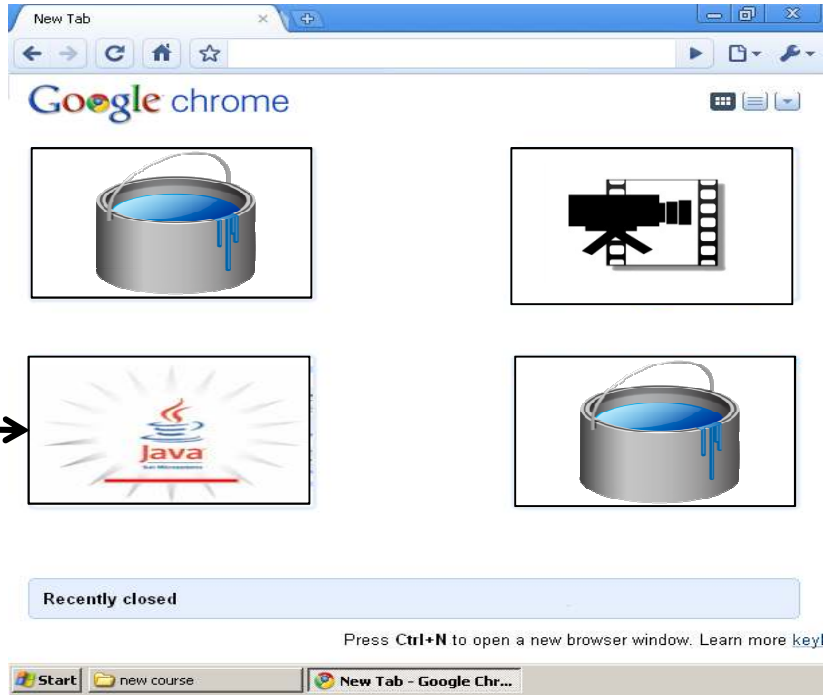
Overview

Web Server

(www.abc.com)

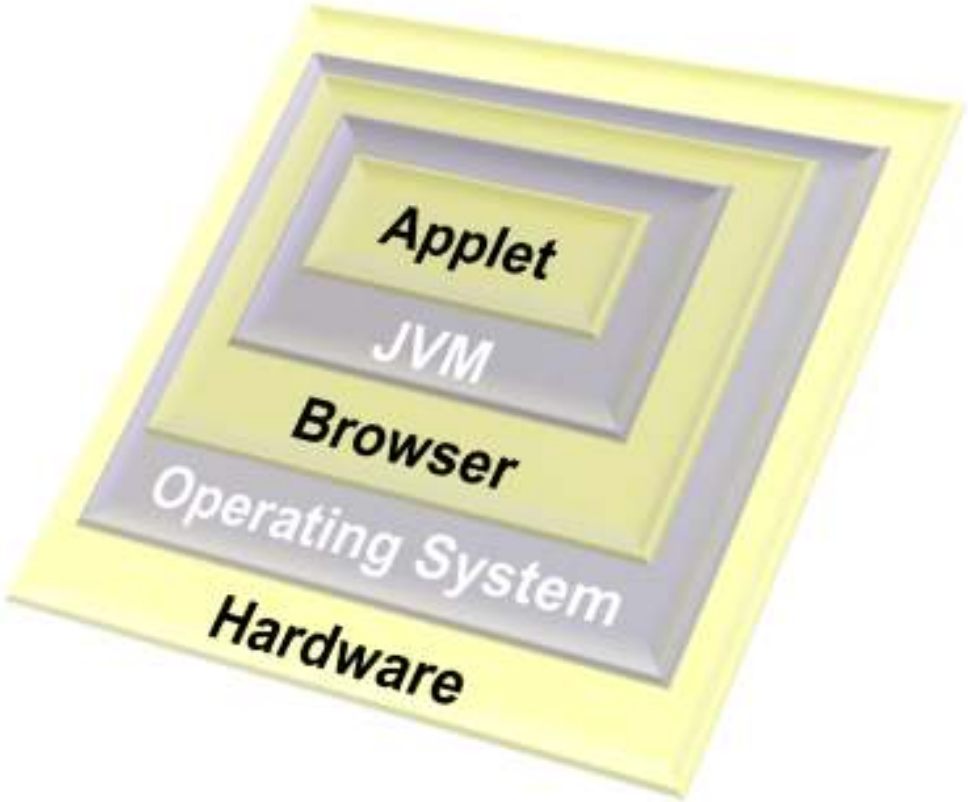
Download MyApplet.class

www.abc.com\index.html



Applet Features

- Machine and Platform Independent



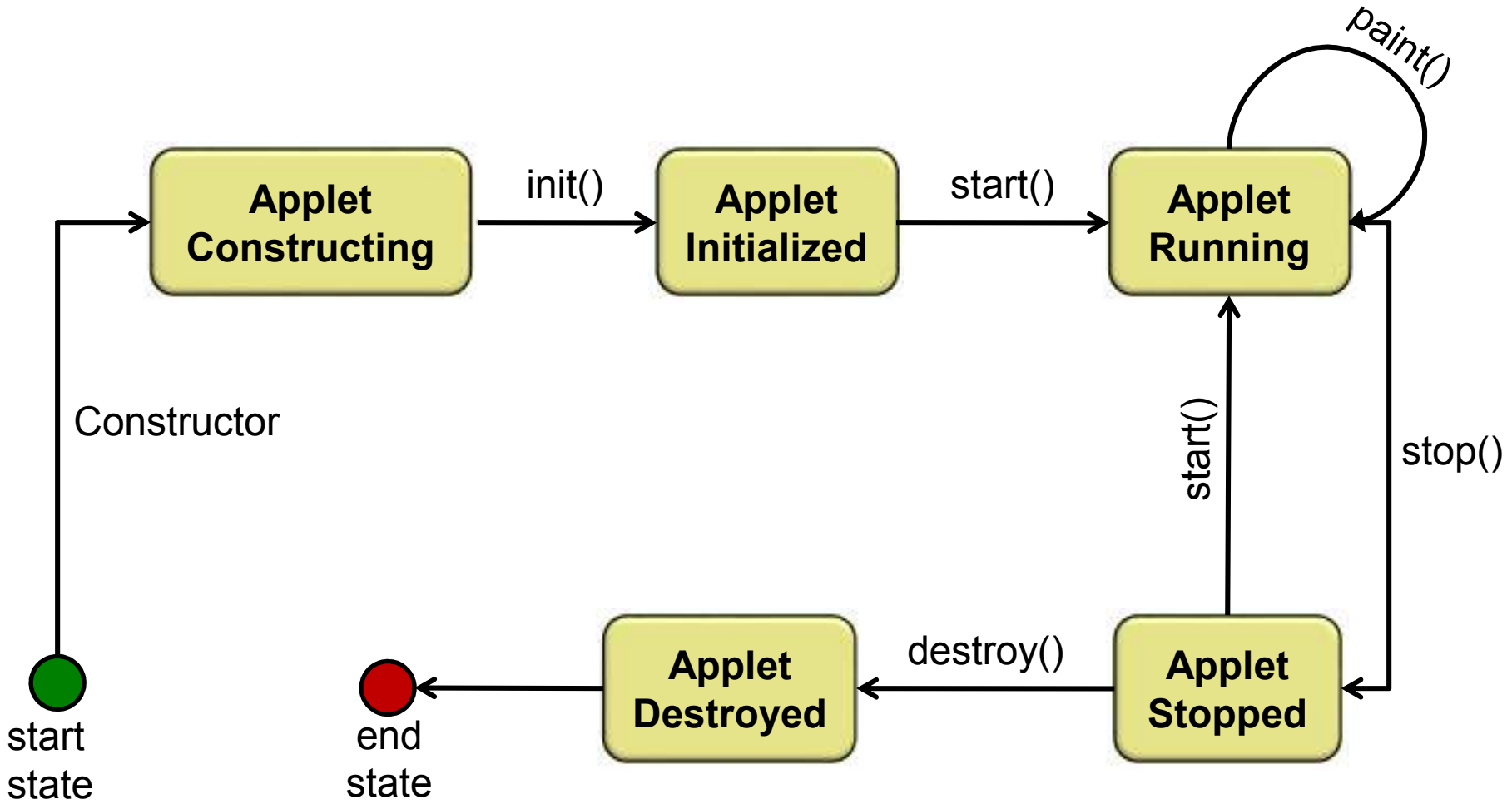
Applets

- 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">
   </applet>
  </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= />`

Lesson 4

Data Types & Operators

Identifiers

- An identifier is the name given to a feature (variable, method, or class).
- An identifier can begin with either:
  - a letter,
  - $, or
  - underscore.
- Subsequent characters may be:
  - a letter,
  - $,
  - underscore, or
  - digits.

Data types

- Data types can be classified into two types:

Primitive | **Reference**

Boolean	`boolean`	1 bit	(true/false)
Integer	`byte`	1 B	(-2^7 → 2^7-1) (-128 → +127)
	`short`	2 B	(-2^{15} → $2^{15}-1$) (-32,768 to +32,767)
	`int`	4 B	(-2^{31} → $2^{31}-1$)
	`long`	8 B	(-2^{63} → $2^{63}-1$)
Floating Point	`float`	4 B	Standard: IEEE 754 Specification
	`double`	8 B	Standard: IEEE 754 Specification
Character	`char`	2 B	unsigned Unicode chars (0 → $2^{16}-1$)

Arrays

Classes

Interfaces

Wrapper Classes

- Each primitive data type has a corresponding wrapper class.

boolean	→	**Boolean**
byte	→	**Byte**
char	→	**Character**
short	→	**Short**
int	→	**Integer**
long	→	**Long**
float	→	**Float**
double	→	**Double**

Wrapper Classes cont'd

- There are three reasons that you might use a wrapper class rather than a primitive:

1. As an argument of a method that expects an object.

2. To use constants defined by the class,
   - such as **MIN_VALUE** and **MAX_VALUE**,
     that provide the upper and lower bounds of the data type.

3. To use class methods for
   - converting values to and from other primitive types,
   - converting to and from strings,
   - converting between number systems (decimal, octal, hexadecimal, binary).

Wrapper Classes cont'd

- They have useful methods that perform some general operation, for example:

`primitive xxxValue()`	→	convert wrapper object to primitive
`primitive parseXXX(String)`	→	convert String to primitive
`Wrapper valueOf(String)`	→	convert String to Wrapper

```
Integer i2 = new Integer(42);
byte b = i2.byteValue();
double d = i2.doubleValue();
```

```
String s3 = Integer.toHexString(254);
System.out.println("254 is " + s3);
```

Wrapper Classes cont'd

- They have special static representations, for example:

`POSITIVE_INFINITY`		In class Float & Double
`NEGATIVE_INFINITY`		
`NaN`	Not a Number	

Literals

- A literal is any value that can be assigned to a primitive data type or String.

boolean	true false	
char	‘a’ …. ’z’ ‘A’ …. ‘Z’	
	‘\u0000’ …. ‘\uFFFF’	
	‘\n’ ‘\r’ ‘\t’	
Integral data type	15	Decimal (int)
	15L	Decimal (long)
	017	Octal
	0XF	Hexadecimal
Floating point data type	73.8	double
	73.8F	float
	5.4 E-70	$5.4 * 10^{-70}$
	5.4 e+70	$5.4 * 10^{70}$

Reference Data types: Classes

- General syntax for creating an object:

```
MyClass myRef;                  // just a reference
myRef = new MyClass();          // construct a new object
```

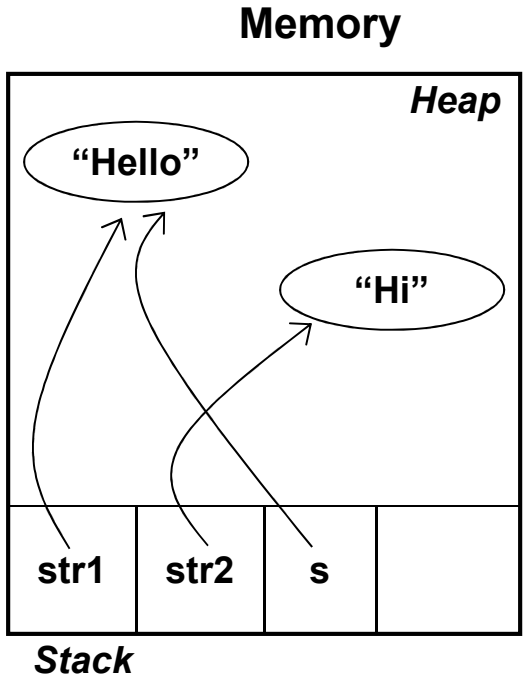
- Or on one line:

```
MyClass myRef = new MyClass();
```

- An object is garbage collected when there is no reference pointing to it.

Reference Data types: Classes cont'd

```
String str1;                          // just a null reference
str1 = new String("Hello");  // object construction

String str2 = new String("Hi");

String s = str1;        //two references to the same object

str1 = null;

s = null;        // The object containing "Hello" is
                 // now eligible for garbage collection.



str1.anyMethod();      // ILLEGAL!
                       //Throws NullPointerException
```



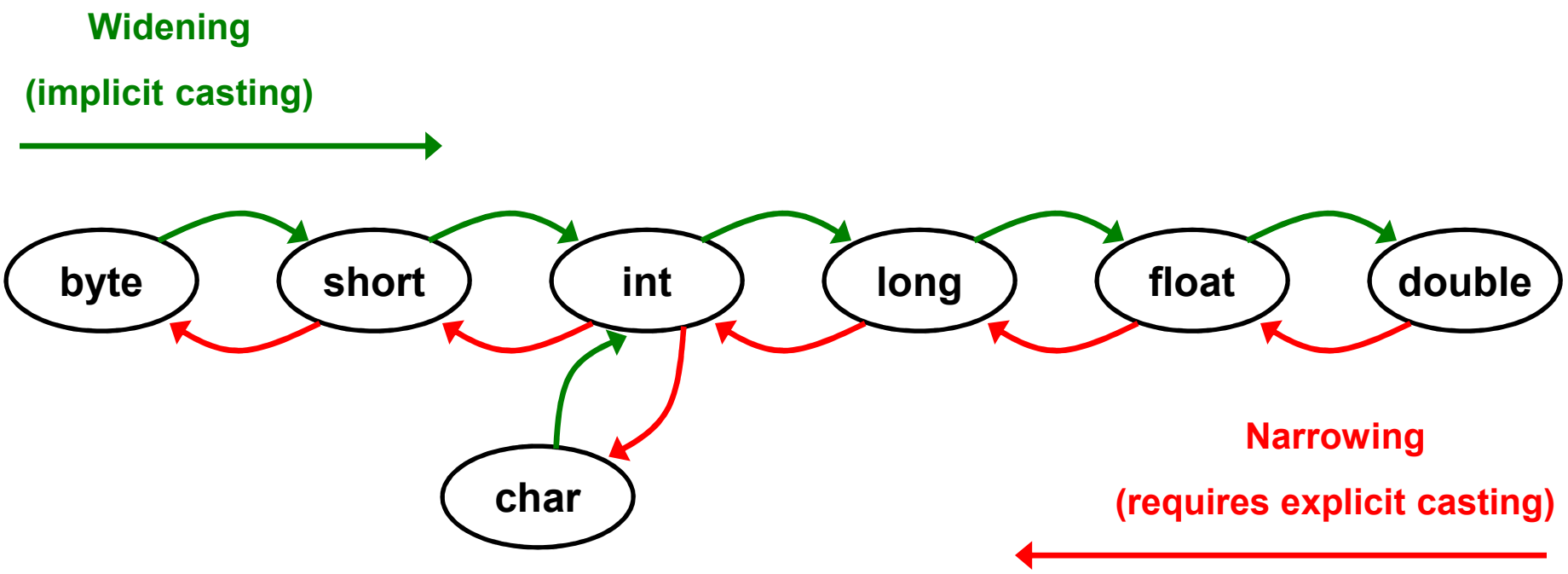
Operators

- Operators are classified into the following categories:
  - Unary Operators.
  - Arithmetic Operators.
  - Assignment Operators.
  - Relational Operators.
  - Shift Operators.
  - Bitwise and Logical Operators.
  - Short Circuit Operators.
  - Ternary Operator.

Operators cont'd

- Unary Operators:

+	-	++	--	!	~	( )
positive	negative	increment	decrement	boolean complement	bitwise inversion	casting



Operators cont'd

- Arithmetic Operators:

+	-	*	/	%
add	subtract	multiply	division	modulo

- Assignment Operators:

=	+=	-=	*=	/=	%=	&=	|=	^=

- Relational Operators:

<	<=	>	>=	==	!=	instanceof

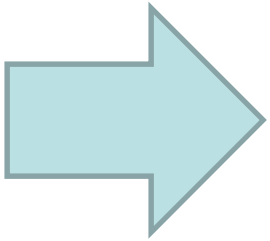
Operations must be performed on homogeneous data types

Operators cont'd

```
byte b=10;
byte b1=15;
byte b2=b+b1;
Value of b2 is ?
```

Compilation Error – Explicit Cast Needed to convert from integer to byte

```
5%2      1
5%-2     1
-5%2     -1
-5%-2    -1
```



```
int x=1234567899
int y=567899999
int z=x*y/x
```

Unexpected results

Operators cont'd

- Shift Operators:

>>	<<	>>>
right shift	left shift	unsigned right shift

- Bitwise and Logical Operators:

&	|	^
AND	OR	XOR

- Short Circuit Operators:

&&	||
(condition1 AND condition2)	(condition1 OR condition2)

Operators cont'd

- Ternary Operator:

```
condition ?true statement:false statement
```

```
int y=15;
int z=12;
int x=y<z?10:11;
```

```
If(y<z)
x=10;
else
x=11;
```

Operators cont'd

Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
Bitwise and Logical AND	&
bitwise exclusive OR	^
Bitwise and Logical inclusive OR	|
Short Circuit AND	&&
Short Circuit OR	||
ternary	? :
assignment	= op=

Lesson 5

Using Arrays & Strings

What is Array?

- An Array is a collection of variables of the same data type.
- Each element can hold a single item.
- Items can be primitives or object references.
- The length of the array is determined when it is created.

What is Array?

- Java Arrays are homogeneous.
- You can create:
  - An array of primitives,
  - An array of object references, or
  - An array of arrays.
- If you create an array of object references, then you can store subtypes of the declared type.

Declaring an Array

- General syntax for creating an array:

```
Datatype[]  arrayIdentifier;              // Declaration
arrayIdentifier = new Datatype [size];  // Construction
```

- Or on one line, hard coded values:

```
Datatype[] arrayIdentifier = { val1, val2, val3, val4 };
```

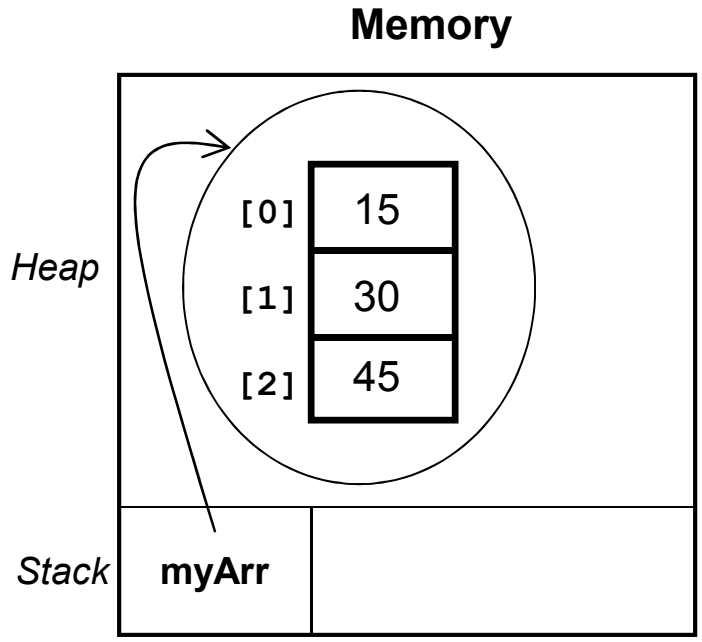
- To determine the size (number of elements) of an array at runtime, use:

```
arrayIdentifier.length
```

Declaring an Array cont'd

- **Example1:** Array of Primitives:

```
int[] myArr;

myArr = new int[3];

myArr[0] = 15 ;
myArr[1] = 30 ;
myArr[2] = 45 ;

System.out.println(myArr[2]);
```

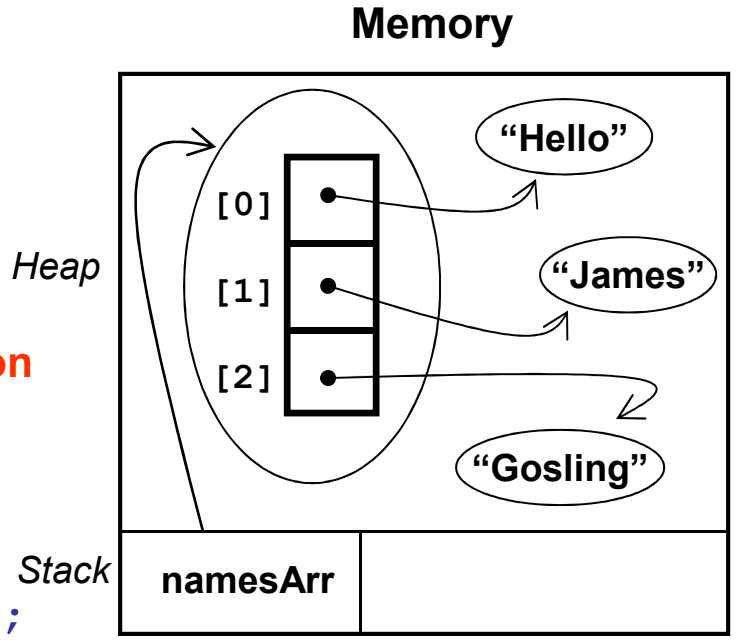



```
myArr[3] = … ; // ILLEGAL!
        //Throws ArrayIndexOutOfBoundsException
```

Declaring an Array cont'd

- **Example2:** Array of Object References:

```
String[] namesArr;

namesArr = new String[3];

namesArr[0].anyMethod()  // ILLEGAL!
                  //Throws NullPointerException

namesArr[0] = new String("Hello");
namesArr[1] = new String("James");
namesArr[2] = new String("Gosling");

System.out.println(namesArr[1]);
```



String Operations

- Although String is a reference data type (class),
  - it may figuratively be considered as the 9th data type because of its special syntax and operations.
  - Creating String Object:

```
String myStr1 = new String("Welcome");
String sp1 = "Welcome";
String sp2 = " to Java";
```

  - Testing for String equality:

```
if(myStr1.equals(sp1))

if(myStr1.equalsIgnoreCase(sp1))

if(myStr1 == sp1)
   // Shallow Comparison (just compares the references)
```

Strings Operations cont'd

- The '+' and '+=' operators were overloaded for class String to be used in concatenation.

```
String str = myStr1 + sp2;          // "Welcome to Java"
str += " Programming";              // "Welcome to Java Programming"
str = str.concat(" Language");      // "Welcome to Java Programming Language"
```

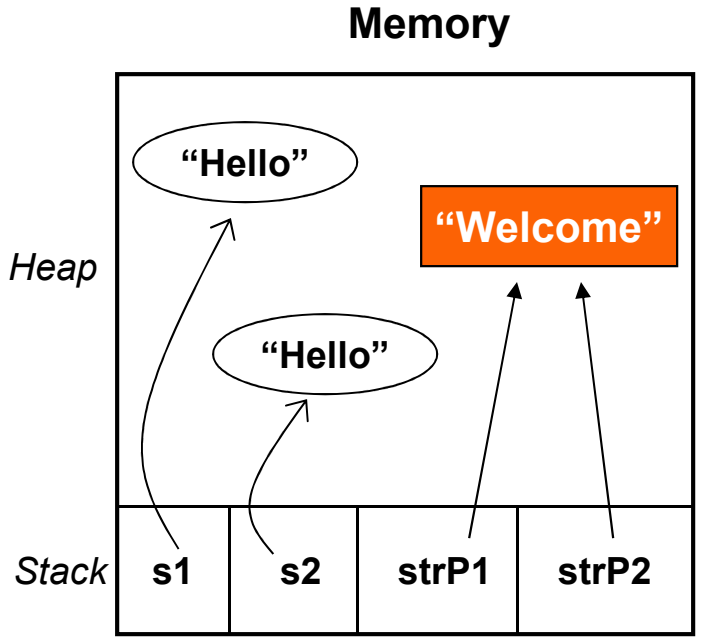
- Objects of class String are immutable
  - you can't modify the contents of a String object after construction.

- Concatenation Operations always return a new String object that holds the result of the concatenation. The original objects remain unchanged.

String Pool

- String objects that are created without using the “new” keyword are said to belong to the “String Pool”.

```
String s1 = new String("Hello");
String s2 = new String("Hello");

String strP1 = "Welcome" ;
String strP2 = "Welcome" ;
```



String Pool cont'd

- String objects in the pool have a special behavior:
  - If we attempt to create a fresh String object with exactly the same characters as an object that already exists in the pool (case sensitive), then no new object will be created.
  - Instead, the newly declared reference will point to the existing object in the pool.
- Such behavior results in a better performance and saves some heap memory.
- Remember: objects of class String are immutable.

Lesson 6

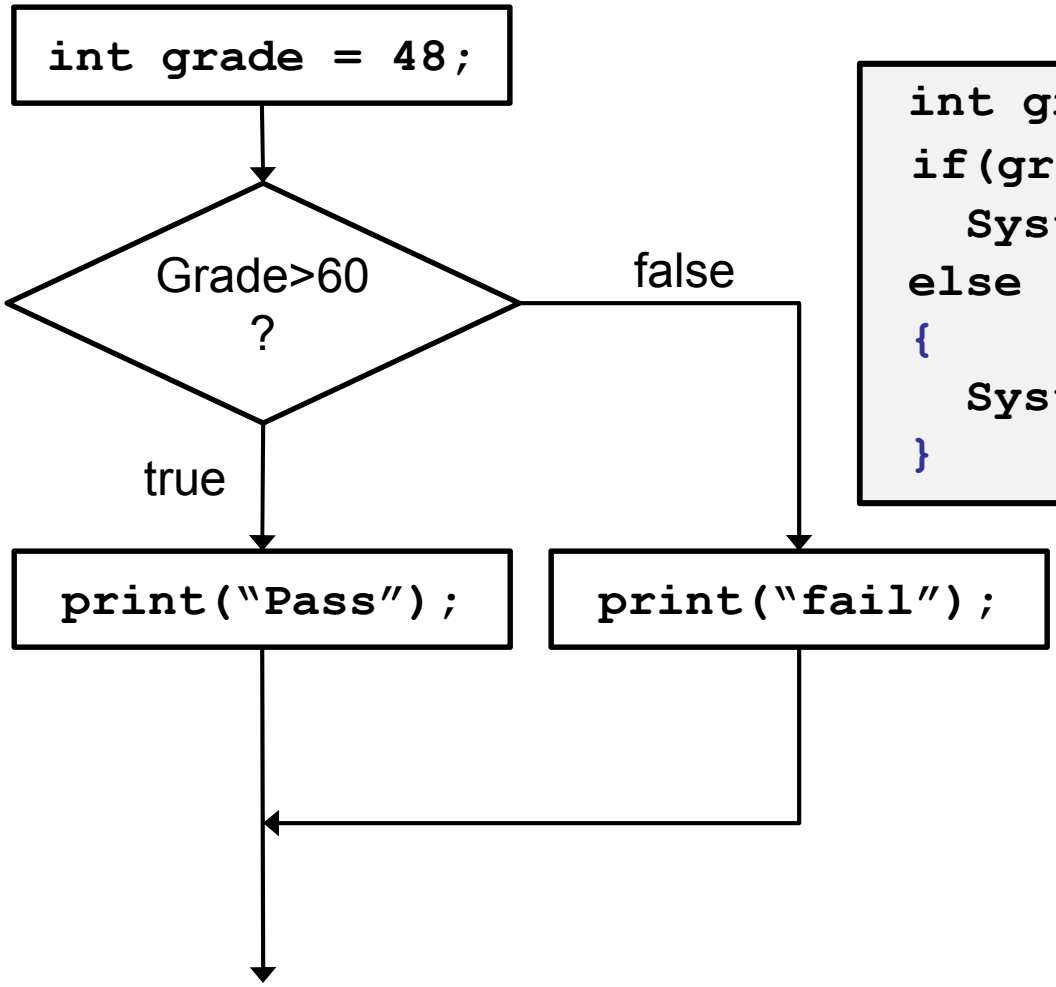
Controlling Program Flow

Flow Control: Branching - if, else

- The if and else blocks are used for binary branching.
- **Syntax:**

```
if(boolean_expr)
{
        …
        …     //true statements
        …
}
[else]
{
        …
        …     //false statements
        …
}
```

if, else Example




```
int grade = 48;
if(grade > 60)
  System.out.println("Pass");
else
{
  System.out.println("Fail");
}
```

Flow Control: Branching - switch

- The switch block is used for multiple branching.
- **Syntax:**

```
switch(myVariable){
      case value1:
         …
         …
      break;
      case value2:
         …
         …
      break;
      default:
         …
}
```

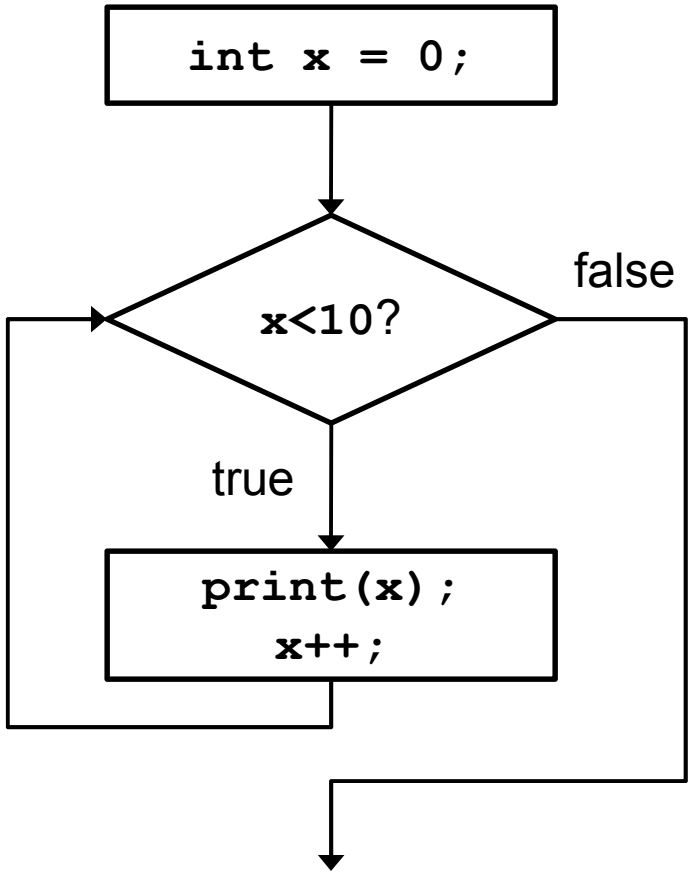
- byte
- short
- int
- char
- enum

Flow Control: Iteration – while loop

- The while loop is used when the termination condition occurs unexpectedly and is checked at the beginning.
- **Syntax**:

```
while (boolean_condition)
{
      …
      …
      …
}
```

while loop Example



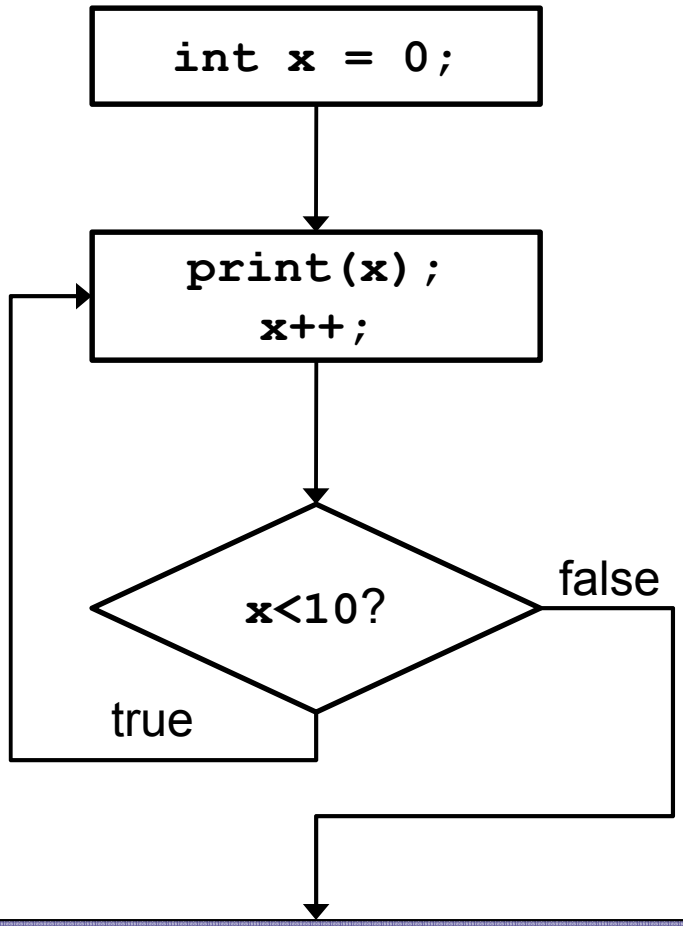

```
int x = 0;
while (x<10) {
     System.out.println(x);
     x++;
}
```

Flow Control: Iteration – do..while loop

- The do..while loop is used when the termination condition occurs unexpectedly and is checked at the end.
- **Syntax:**

```
do
{
        …
        …
        …
}
while(boolean_condition);
```

do..while loop Example




```
int x = 0;
do{
     System.out.println(x);
     x++;
} while (x<10);
```

Flow Control: Iteration – for loop

- The for loop is used when the number of iterations is predetermined.
- **Syntax:**

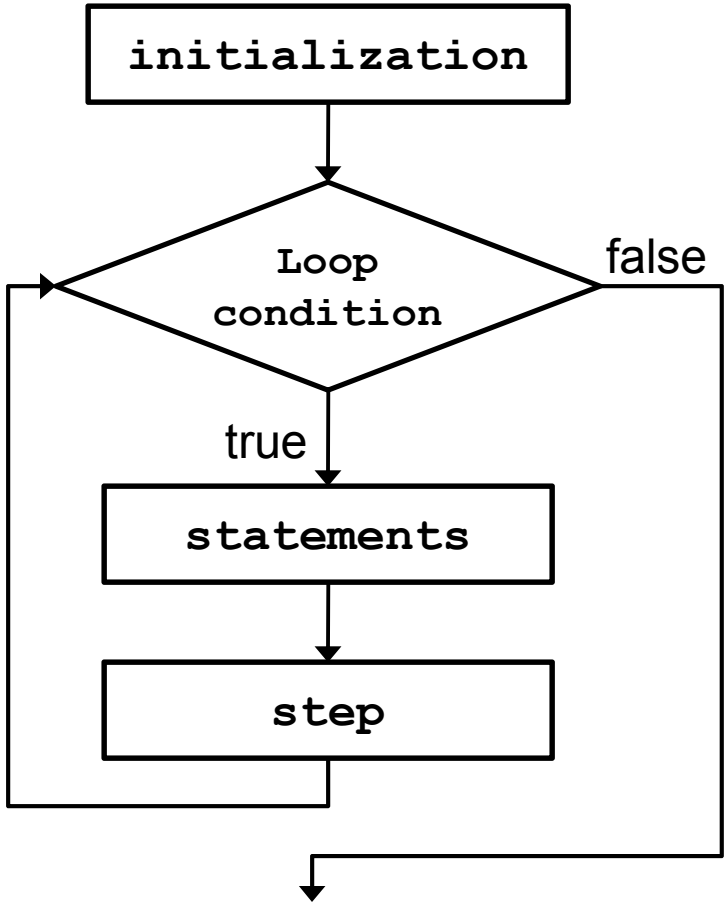
```
for (initialization ; loop_condition ; step)
{
        …
        …
        …
}
```

```
for (int i=0 ; i<10 ; i++)
{
        …
        …
}
```

- You may use the **`break`** and **`continue`** keywords to skip or terminate the iterations.

Flow Control: Iteration – for loop

```
for (initialization ; loop_condition ; step)
```



Flow Control: Iteration –Enhanced for loop

```
for (type identifier : iterable_expression)
{
        // statements
}
```

- **The first element:**
  - is an identifier of the same type as the iterable_expression
- **The second element:**
  - is an expression specifying a collection of objects or values of the specified type.
- The enhanced loop is used when we want to iterate over arrays or collections.

Flow Control: Iteration –Enhanced for loop example

```
double[] samples = new double[50];
```

```
double average = 0.0;
for(int i=0;i<samples.length;i++)
{
     average += samples[i];
}

average /= samples.length;
```

```
double average = 0.0;
for(double value : samples)
{
     average += value;
}
average /= samples.length;
```

The `break` statement

- The break statement can be used in loops or switch.
- It transfers control to the first statement after the loop body or switch body.

```
.......
while(age <= 65)
{
      balance = payment * 1;
      if (balance >= 25000)
            break;
}
.......
```

The `continue` statement

- The continue statement can be used Only in loops.
- Abandons the current loop iteration and jumps to the next loop iteration.

```
.......
for(int year=2000; year<= 2099; year++){
      if (year % 100 == 0)
            continue;
}
.......
```

Comments in Java

- To comment a single line:

```
// write a comment here
```

- To comment multiple lines:

```
/*  comment line 1
    comment line 2
     comment line 3 */
```

Printable Class Documentation (Javadoc)

- To write professional class, method, or variable documentation:

```
/** javadoc line 1
    javadoc line 2
    javadoc line 3 */
```

  - You can then produce the HTML output by typing the following command at the command prompt:

```
javadoc myfile.java
```

Printable Class Documentation (Javadoc)

- The **`Javadoc`** tool parses tags within a Java doc comment.
- These doc tags enable you to
  - auto generate a complete, well-formatted API documentation from your source code.
- The tags start with (@).
- A tag must start at the beginning of a line.

Example of Javadoc

- Example 1:

```
/**
 * @author khaled
 */
```

- Example 2:

```
/**
 * @param args the command line arguments
 */
```

Lab Exercise

1. Command Line Calculator

- Create a simple non-GUI Application that carries out the functionality of a basic calculator (addition, subtraction, multiplication, and division).

- The program, for example, should be run by typing the following at the command prompt:

  java Calc 70 + 30

2. String Separator

- Create a non-GUI Application that accepts a well formed IP Address in the form of a string and cuts it into separate parts based on the dot delimiter.

- The program, for example, should be run by typing the following at the command prompt:

  java IPCutter 163.121.12.30

- The output should then be:

  163
  121
  12
  30

- Write a program that print the following patterns:

1.
```
*
**
***
****
*****
******
```

2.