Concept of Core & Advance Java

(MCS-304)

(MSc Computer Science Semester 3)

Gurmeet Singh

Assistant Professor, PG Deptt. Of C.Sc & IT

Hans Raj Mahila Maha Vidyalaya, Jalandhar

gurmeethmv@gmail.com

Syllabus

- **Java Fundamentals:** Features, Objects Oriented Basis, Java Virtual Machine Character Set, Operators, Data Types, Control Structures.
- Classes, Inheritance, Polymorphism, Packages & Interfaces.
- Stream IO Classes, Exception Handling.
- **Multithreading:** Java Thread model, Thread Priorities, Synchronization, Interthread communication, Suspending, resuming & stopping thread.
- **Applet:** Applet basics, Applet architecture, Applet: Display, Repaint, Parameter Passing.
- Telnet, FTP, Web Server and their implementation in Java.

Different Programming Paradigms

- Functional/procedural programming:
  - program is a list of instructions to the computer
- Object-oriented programming
  - program is composed of a collection *objects that communicate with each other*

Main Concepts

- Object
- Class
- Inheritance
- Encapsulation

Objects

- identity – unique identification of an object
- attributes – data/state
- services – methods/operations
  - supported by the object
  - responsibility to provide services to other clients

Class

- “type”
- object is an **instance** of class
- class groups similar objects
  - same (structure of) attributes
  - same services
- object holds values of its class’s attributes

Inheritance

- Class hierarchy
- Generalization and Specialization
  - subclass inherits attributes and services from its superclass
  - subclass may add new attributes and services
  - subclass may reuse the code in the superclass
  - subclasses provide specialized behaviors (overriding and dynamic binding)
  - partially define and implement common behaviors (abstract)

Encapsulation

- Separation between internal state of the object and its external aspects
- How ?

- Modularity
  - source code for an object can be written and maintained independently of the source code for other objects
  - easier maintenance and reuse
- Information hiding
  - other objects can ignore implementation details
  - security (object has control over its internal state)

Why Java ?

- Portable
- Easy to learn

- [Designed to be used on the Internet]

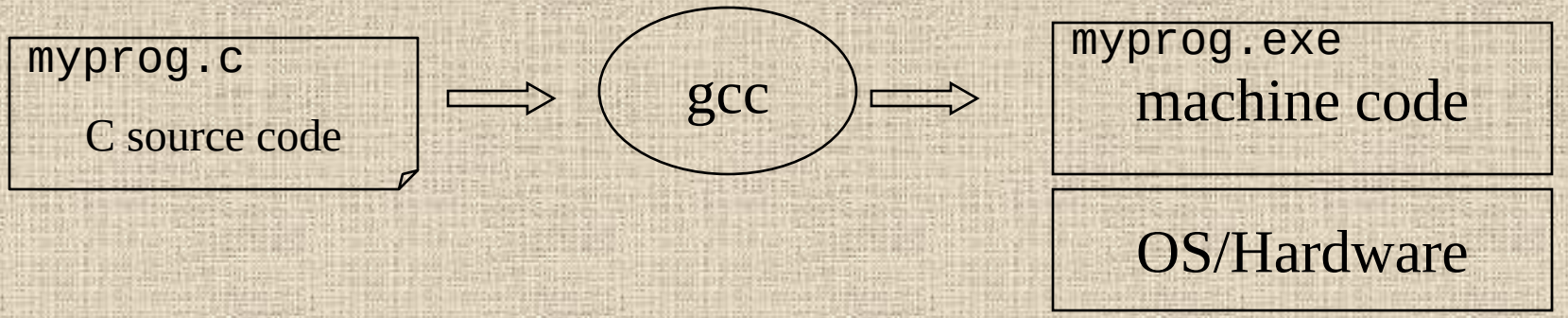
JVM

- JVM stands for

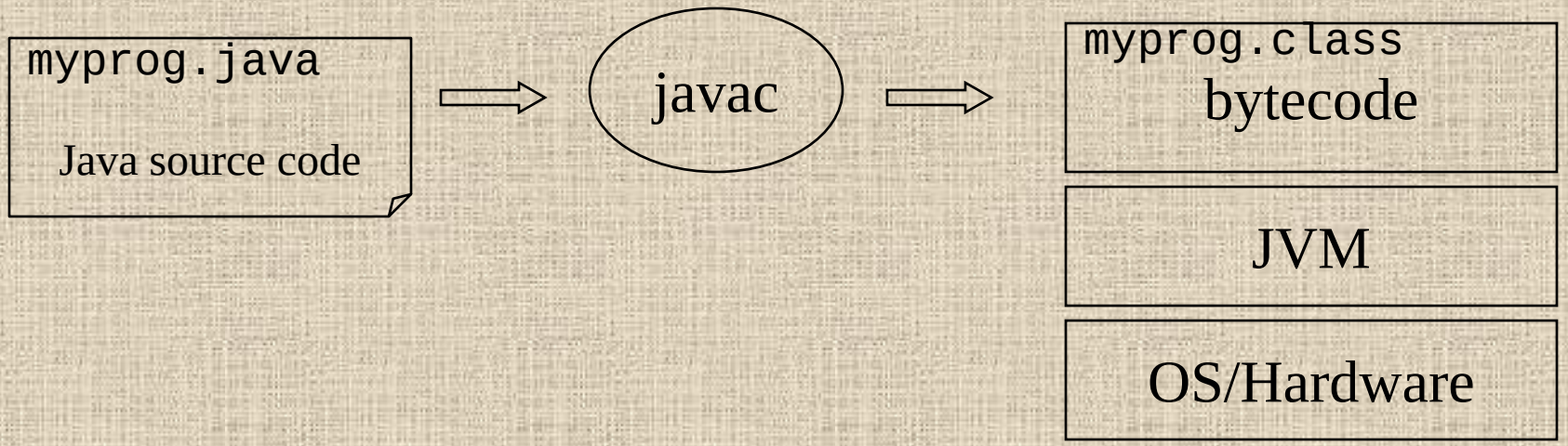
  Java **V**irtual **M**achine

- Unlike other languages, Java “executables” are executed on a CPU that does not exist.

Platform Dependent



Platform Independent



Character Set

UNICODE character set (2 byte) is used to store per character.

Disadvantages of ASCII code set??

Data Types

- int 4 bytes
- short 2 bytes
- long 8 bytes
- byte 1 byte
- float 4 bytes
- double 8 bytes
- char Unicode encoding (2 bytes)
- boolean {true,false}

Behaviors is exactly as in C++

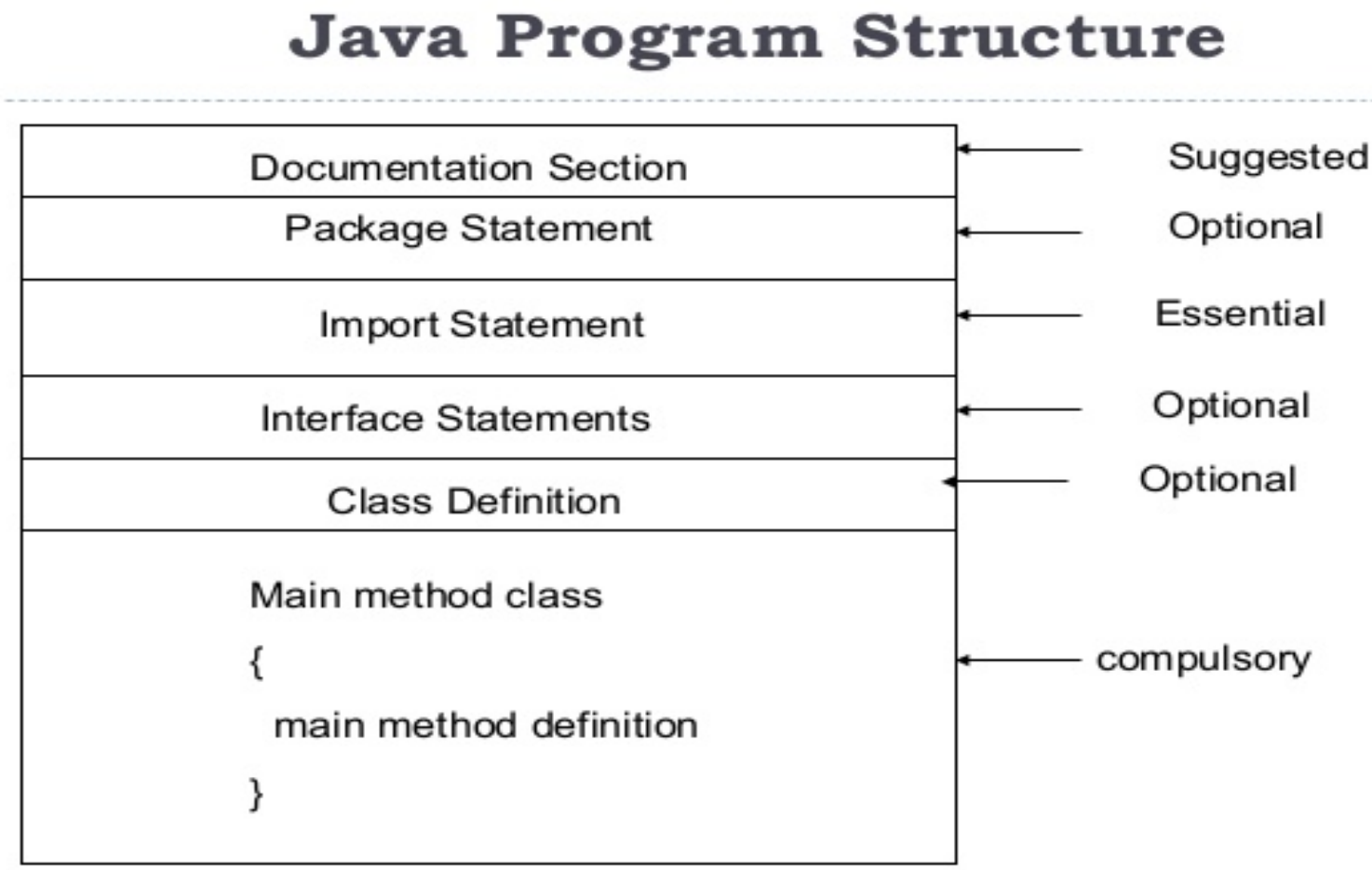
Note:
Primitive type always begin with lower-case

Literals

- **Constants**

37	integer
37.2	float
42F	float
0754	integer (octal)
0xfe	integer (hexadecimal)

Structure of JAVA Program



Hello World

Hello.java

```
class Hello {
      public static void main(String[] args) {
            System.out.println(“Hello World !!!”);
      }
}
```

C:\javac Hello.java *(compilation creates Hello.class)*

C:\java Hello *(Execution on the local JVM)*

Operators

- Arithmetic operators
- Relation operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators
- Misc operators

Arithmetic Operators

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denumerator
%	remainder of division
++	Increment operator increases integer value by one
--	Decrement operator decreases integer value by one

Relational Operators

Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

Logical Operators

Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	left shift
<<	right shift

Assignment Operators

Operator	Description	Example
=	assigns values from right side operands to left side operand	a = b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

Other Operators

- ✓ Conditional Operator
- ✓ instanceOf Operator
- ✓ String Concatenation Operator
- ✓ Dot Operator
- ✓ new Operator
- ✓ Cast Operator (type)

Flow control

Exactly like c/c++.

if/else

```
If(x==4) {
  // act1
} else {
  // act2
}
```

do/while

```
int i=5;
do {
  // act1
  i--;
} while(i!=0);
```

for

```
int j;
for(int i=0;i<=9;i++)
{
  j+=i;
}
```

switch

```
char
c=IN.getChar();
switch(c) {
  case ‘a’:
  case ‘b’:
    // act1
    break;
  default:
    // act2
}
```

Flow control

- ✓While loop
- ✓break & labelled break
- ✓Continue & labelled continue

Arrays

- ✓ Array is an object
- ✓ Array size is fixed

Arrays - Multidimensional

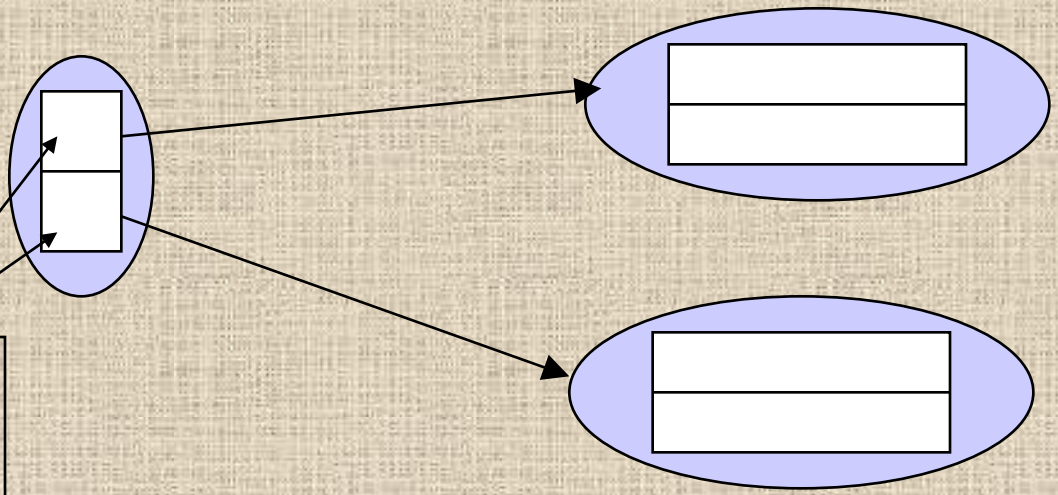
- In C++

```
Animal arr[2][2]
```

Is:

- In Java

```
Animal[][] arr=
   new Animal[2][2]
```



What is the type of the object here ?

Static Data Members

- **Member data** - Same data is used for all the instances (objects) of some Class.

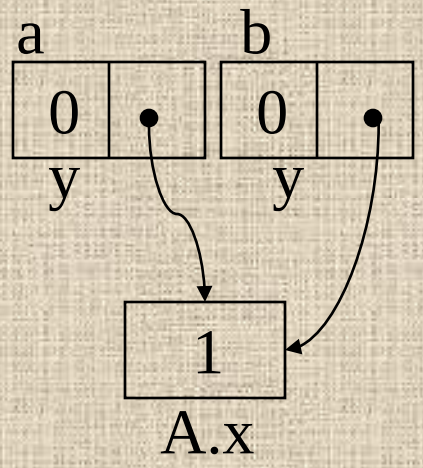
```
Class A {
   public int y = 0;
   public static int x = 1;
};

A a = new A();
A b = new A();
System.out.println(b.x);
a.x = 5;
System.out.println(b.x);
A.x = 10;
System.out.println(b.x);
```

Assignment performed on the first access to the Class.
Only one instance of 'x' exists in memory

Output:

```
1
5
10
```



Static Functions

- **Member function**
  - Static member function can access only static members
  - Static member function can be called without an instance.

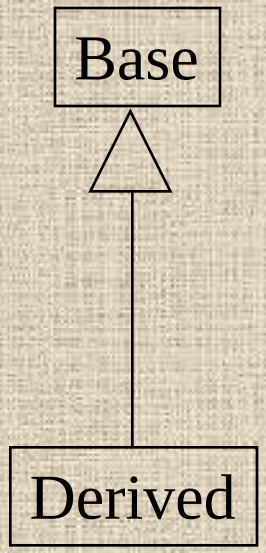
```
Class Pots {
        private static int num = 0;
        private Color clr_;
        public Pot(Color c) {
                clr = c;
                num++;
        }
        public static int Num_Of_Pots()
                { return num; }

        // error :
        public static Color getColor()
                { return clr; }
}
```

Access Control

- ***public*** member (function/data)
  - Can be called/modified from outside.
- ***protected***
  - Can be called/modified from derived classes
- ***private***
  - Can be called/modified only from the current class
- ***default (if no access modifier stated)***
  - Usually referred to as “Friendly”.
  - Can be called/modified/instantiated from the same package.

Inheritance




```
class Base {
    Base(){}
    Base(int i) {}
    protected void foo() {…}
}

class Derived extends Base {
    Derived() {}
    protected void foo() {…}
    Derived(int i) {
      super(i);
       …
      super.foo();
    }
}
```

Polymorphism

- Inheritance creates an “kind of” relation:

For example, if B inherits from A, than we say that “B is kind of A”.

  - access rights (Java **forbids** reducing access rights)

Inheritance

- In Java, all methods are virtual :

```
class Base {
  void foo() {
    System.out.println(“Base”);
  }
}
class Derived extends Base {
  void foo() {
    System.out.println(“Derived”);
  }
}
public class Test {
  public static void main(String[] args) {
    Base b = new Derived();
    b.foo();  // Derived.foo() will be activated
  }
}
```

Inheritance

```
class classC extends classB {
   classC(int arg1, int arg2){
     this(arg1);
     System.out.println("In classC(int arg1, int arg2)");
   }
   classC(int arg1){
     super(arg1);
     System.out.println("In classC(int arg1)");
   }
}
class classB extends classA {
   classB(int arg1){
     super(arg1);
     System.out.println("In classB(int arg1)");
   }
   classB(){
     System.out.println("In classB()");
   }
}
```

Inheritance

```
class classA {
   classA(int arg1){
        System.out.println("In classA(int arg1)");
   }
   classA(){
     System.out.println("In classA()");
   }
}

class classB extends classA {
   classB(int arg1, int arg2){
     this(arg1);
     System.out.println("In classB(int arg1, int arg2)");
   }
   classB(int arg1){
     super(arg1);
     System.out.println("In classB(int arg1)");
   }

  class B() {
    System.out.println("In classB()");
  }
}
```

Abstract

- ***abstract*** member function, means that the function does not have an implementation.
- ***abstract*** class, is class that can not be instantiated.

```
AbstractTest.java:6: class AbstractTest is an abstract class.
It can't be instantiated.
        new AbstractTest();
        ^
1 error
```

NOTE:
An abstract class is not required to have an abstract method in it.
But any class that has an abstract method in it or that does
not provide an implementation for any abstract methods declared
in its superclasses must be declared as an abstract class.

Abstract - Example

```
package java.lang;
public abstract class Shape {
       public abstract void draw();
       public void move(int x, int y) {
         setColor(BackGroundColor);
         draw();
         setCenter(x,y);
         setColor(ForeGroundColor);
         draw();
        }
}
```

```
package java.lang;
public class Circle extends Shape {
       public void draw() {
        // draw the circle ...
      }
}
```

Interface

Interfaces are useful for the following:

- Capture similarities among unrelated classes without forcing a class relationship.
- Declare methods that one or more classes are expected to implement.
- Java Event Handling is through Interfaces.

Interface

- abstract “class”
- All methods are public
- Classes need to implement the interfaces.
- All functions needs to be implemented by the class, otherwise abstract.

Interface

```
interface I1 {
   void cook(Food food);
}
```

```
interface I2 {
   void kick(Football);
}
```

```
interface I3 {
   void curse();
}
```

```
class A implements I1, I3 {
      // overridden methods MUST be public
      // can you tell why ?
      public void curse() { … }
      public void cook(Food f) { … }
}
```

final

- ***final* member data**
  Constant member
- ***final* member function**
  The method can't be overridden.
- ***final* class**
  'Base' is final, thus it can't be extended

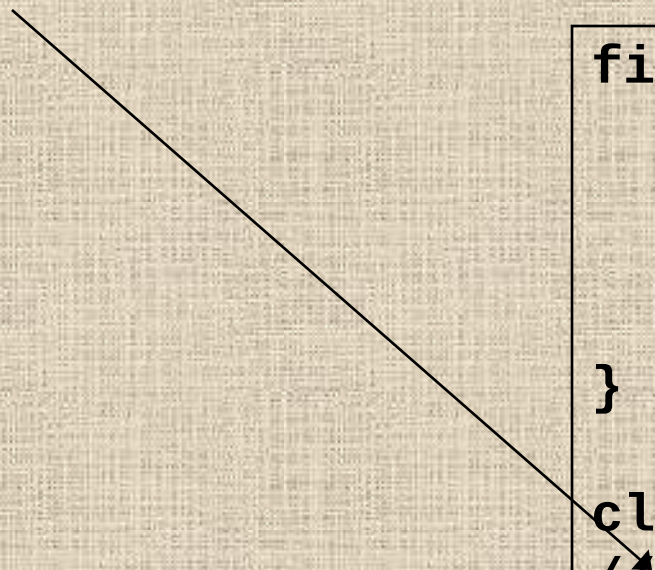
```
final class Base {
  final int i=5;
  final void foo() {
     i=10;
//what will the compiler say
about this?
   }
}

class Derived extends Base {
// Error
   // another foo ...
   void foo() {

   }
}
```

final

```
Derived.java:6: Can't subclass final classes: class Base
class class Derived extends Base {
      ^
1 error
```



```
final class Base {
  final int i=5;
  final void foo() {
    i=10;
  }
}

class Derived extends Base {
// Error
  // another foo ...
  void foo() {

  }
}
```

String Handling Functions

- ✓ length()
- ✓ charAt()
- ✓ concat()
- ✓ indexOf()
- ✓ lastIndexOf()
- ✓ equals(String)
- ✓ equalsIgnoreCase(String)
- ✓ compareTo(String)
- ✓ startsWith(String)

String Handling Functions

- ✓ endsWith(String)
- ✓ toLowercase()
- ✓ toUppercase()
- ✓ trim()
- ✓ replace()
- ✓ replaceAll()
- ✓ replaceFirst()
- ✓ split()
- ✓ matches()

String Handling Functions

- ✓ substring(int)
- ✓ substring(int,int)
- ✓ toCharArray()
- ✓ getChars()
- ✓ valueOf()

StringBuffer Functions

- ✓ length()
- ✓ capacity()
- ✓ ensureCapacity()
- ✓ setLength(int)
- ✓ charAt(int)
- ✓ subString(int)
- ✓ subString(int,int)
- ✓ setCharAt(int,char)
- ✓ reverse()

StringBuffer Functions

- ✓ toString()
- ✓ append(String)
- ✓ append(char[], int, int)
- ✓ insert(int,String)
- ✓ insert(int,char[],int,int)
- ✓ delete(int,int)
- ✓ deleteCharAt(int)