# Java

OBJECT ORIENTED PROGRAMMING LANGUAGE

WRITE ONCE, RUN ANYWHERE

# Objective



- To learn the basics of OOPS
- To learn basics of Core Java
- To learn JDBC
- To learn problem solving and code analysis

# Introduction





Java

- ✓ Developed by Sun Microsystems.
- ✓ Java 1.0 in 1995
- ✓ Guaranteed to be write once, run anywhere

Java Editions:

- ➢ J2SE
- ➢ J2EE
- ➢ J2ME

# Java is:



- Object Oriented
- Platform Independent
- Simple
- Secure
- Portable
- Architectural Neutral
- Robust
- Multi Threaded
- Interpreted
- High-Performance
- Distributed

# Java & C++ — Similarities & Differences



Java resembles C++ only superficially:

- Similarities:
  - primitive data types (in Java, platform independent)
  - syntax: control structures, exceptions ...
  - classes, visibility declarations (public, private)
  - multiple constructors, this, new
  - types, type casting

# Contd…



- Extensions:
  - garbage collection
  - run time type information
  - standard classes (Strings, collections …)
  - packages
  - final classes

# Java and C++ - Simplifications



Java eliminates many of the complex features of C++ :

- no pointers — just references
- no functions — can declare static methods
- no global variables — can declare public static variables
- no destructors — garbage collection and finalize methods
- no linking — dynamic class loading
- no operator overloading — only method overloading
- no member initialisation lists — super constructor can be called
- no pre-processor — static final constants
- no multiple inheritance — multiple interfaces
- no structs, unions, enums — typically not needed

# What is JIT?



- Just-in-Time Compilation
- Compiles the Byte code to platform specific executable code making the execution faster.
- JIT compilation can be treated as second compilation. First compilation is converting the source code to platform independent Byte code.

- Generally,  Byte code is interpreted by JVM.

# What is Java Byte code?



- Byte code is platform independent code.
- Java compiler converts the source code to Byte code.
- As Byte code is platform independent, code once written in one platform can be executed in any platform that supports JVM.

- This is the reason Java is platform independent and Architecture neutral.

# OOP Principles





- Abstraction (Abstract Methods, Objects)
- Encapsulation (Objects)

Abstraction and Encapsulation are closely tied together

- Inheritance (Class inheritance)
- Polymorphism (Method overloading)

# Basics Terminology



- Class- Template/blueprint that defines the behavior/states of its objects.
- Object- Instance of a class, have states and behavior
- Methods-Basically a behavior
- Instance Variables-Members of class

# Java First Program



```
public class MyFirstJavaProgram{

/* This is my first java program.
    * This will print 'Hello World' as the output
    */

  public static void main(String[]args){
System.out.println("Hello World");// prints Hello World
}
}
```



Vamsee Krishna Kiran M, Asst.Prof, Dept. of CSE, AVVP

# Why **static** keyword before main() ?



- Methods that are declared **static** doesn't need instantiation.
- Therefore methods can be invoked as follows:

Classname.methodName();

# What is System.out.println();



- **System** is a class in the **java**.lang package. **out** is a static member of the **System** class, and is an instance of **java**.io.PrintStream . **println** is a method of **java**.io.PrintStream.

- Here **println** function is overloaded to write the output stream to either console or file.

# Which package is always imported default in Java?



- Java.lang
- Lang package contains some important classes like
  1. System
  2. Object
  3. String
  4. String builder, String Buffer
  5. Process
  6. Thread
  7. Exception
  8. Math
  9. All Wrapper types like Integer, Float, Character, Boolean etc.

# What is the difference between Class variable and Instance variable?







- Every object has it's own copy of the instance variables.

Ex:
```
public class Vehicles
{
    public int count;
    /*...*/
}
```

- Class variable also known as static member variables, only have **one** copy of the variable(s) shared with all instances of the class.

Ex:
```
public class Vehicles
{
    public static int count;
    /*...*/
}
```

# Which items in Java occupies stack and which one's occupy heap?





## Heap

- Class variables
- Instance variables
- Objects
- Strings
- Arrays
- Static methods and variables

## Stack

- Local variables or method variables
- All primitive types
- Methods references are stored on stack, though methods variables stay on heap

Now,
How many stacks will be created for each program?

# What is an Abstract class?



- Abstract class is a class that is declared abstract
- Prefix the keyword **abstract** before the class name
- Contains both abstract methods as well as concrete methods
- Abstract classes cannot be instantiated
- So, how will you use this abstract class?
  - By deriving from abstract class
  - Derived class has to provide definitions for all the abstract methods.
  - What if it doesn't provide definitions for all?
- What happens if a class includes abstract methods?

# What is an interface?



- It is a collection of abstract methods
- Contains only abstract methods
- All fields declared inside interface are by default public static final
- An interface can extend multiple interfaces
- An interface is not **extended** by a class it is **implemented**
- Interfaces can be written in a file with .java ext, and the name of the interface has to match with name of the file.
- Interfaces cannot be instantiated and doesn't contain constructors.

# What is the difference between abstract class and interface?







## Abstract class

- You can declare fields that are not static and final
- Contains both abstract methods and concrete methods
- Derived class can extend only one abstract class

## Interface

- All fields are public, static and final
- All methods are by default abstract
- Derived class can implement any number of interfaces

# What is a package?







- A package is a namespace that organizes a set of related classes and interfaces.
- It keeps things organized by placing the related classes and interfaces into packages just like folders in our desktop.
- Prevents naming conflicts and provides namespace management.
- Since the package creates a new namespace there won't be any name conflicts with names in other packages.
- Programmers can define their own packages.

Syntax:

```
Package Vehicles;        // this should be the first line in the source code.
```

# What are the different Access specifiers?



| Access Modifiers | Default | Private | Protected | Public |
|---|---|---|---|---|
| Accessible inside the class | Yes | Yes | Yes | Yes |
| Within the subclass inside the same package | Yes | No | Yes | Yes |
| Outside the package | No | No | No | Yes |
| Within the subclass outside the same package | No | No | Yes | Yes |

# What is the significance of **new** keyword in creating an object?





Vamsee Krishna Kiran M, Asst.Prof, Dept. of CSE, AVVP

- New keyword is responsible for allocating memory for the objects.
- When compiler comes across the **new** keyword it allocates memory for that object in the heap.

Cars objBMW=new Cars(); // this statement allocates objBMW memory in heap.

Above statement can be written as

Cars objBMW; // this is creating the reference object for Cars class.

objBMW=new Cars(); // this statement actually allocates memory for objBMW object.

# Why don't we call **new** keyword for primitive types?



- Primitive types are not objects, therefore they are not created using **new**.
- int, float, char, double, long all these are primitive types or also called value types. They are created just by declaring a variable of that type.

- Integer, Float, Character, Boolean etc. are of class types.

# Which is the super class of all Java classes?



- Every class in Java directly or indirectly derives from the base class **Object** and it is contained in java.lang package.
- Every class inherits all the methods of Object class.
- General functions like
  - getClass();
  - equals();
  - toString();
  - clone();
- Synchronization related functions like
  - wait();
  - notify();
  - notifyAll();
  - finalise();

# What is a wrapper class?







- Wrapper classes are used to wrap a primitive type in an object.
- This action of converting primitive type to wrapper type is called boxing.

- Ex: Integer class, Boolean class, Character class, Float class etc.

Ex: we can create an Integer object which wraps the int 34

```
Integer intObject = new Integer (34);
Integer intObject = new Integer ( "34");
```

Retrieving the value wrapped by a wrapper class object

```
int x = intObject.intValue();
```

# What is a constructor? How is it different from other methods?







- **Constructor in java** is a *special type of method* that is used to initialize the object.
- Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.
- Rules:
  - Constructor name must be same as its class name
  - Constructor must have no explicit return type
- Types:
  - Default constructor (no-arg constructor)
  - Parameterized constructor

```
class Bike{
Bike(){System.out.println("Bike is created");} //default constructor
public static void main(String args[]){
Bike1b=new Bike();
}
}

class Vehicles{
    int id;
    String name;
    Vehicles(int i,String n){         //parameterized constructor
    id = i;  name = n;
    }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    Vehicles  s1 = new  Vehicles(111,"BMW");
    s1.display();
      }
}
```

# What is the significance of **final** keyword in Java?



- **Final** keyword in java can be used in different ways.
- **static final** variable can be used like a constant value.

  Syntax: public static final double PI=3.141;
- **Final** keyword used on a method indicates that the method cannot be overridden by any subclass.
- Declaring a class to be **final** means no subclass can be derived from it.

# Why do you declare a class **final**?



- Adds security to the class.

Ex: By making String class final, you cannot override its length() method.

Also Makes the JVM execution faster.

# Why there is no **virtual** keyword in Java like in C++ and C# ?



- In C++, programmer has to declare **virtual** keyword explicitly to make a method virtual.
- In Java by default all the methods are virtual.
- This allows us to override every method in Java.
- If you want to restrict method overriding, you have to declare that method as **final.**

# How are parameters passed in java, by value or by reference?





- Primitive types are passed by value
- Objects are passed by passing an object handle by value.
- An object handle is nothing but a reference.

Syntax:

```
Vehicles objVeh;
//.....................
SomeClass.method1(objVeh);
```

//here objVeh is a object handle passed to a method of some other class.

Params of methods are always passed by Value.

# How do you call a method in a super class from within a subclass?



- Using the **super** keyword. It is used to refer to immediate parent class variables or methods.

Syntax:

```
class Vehicle{
  int speed=50;
}
class Bike extends Vehicle{
  int speed=100;
  void display(){
   System.out.println(speed + super.speed);
//will print speed of Bike + speed of vehicle
  }
  public static void main(String args[]){
   Bike b=new Bike();
   b.display();
}
}
```

# What is auto boxing and auto unboxing?



- Converting primitive types to object or wrapper type is called as boxing.
- Converting object or wrapper to primitive type is called unboxing.

Syntax:

```
List<Integer> li = new ArrayList<>();
for (int i = 1; i < 50; i += 2)
   li.add(i);
```

Compiler Converts the above code as:

```
List<Integer> li = new ArrayList<>();
for (int i = 1; i < 50; i += 2)
   li.add(Integer.valueOf(i));       //compiler performs autoboxing
```

```
public static int sumEven(List<Integer> li) {
    int sum = 0;
    for (Integer i: li)
        if (i % 2 == 0)
            sum += i;
        return sum;
}
```

Compiler converts the above code as:

```
public static int sumEven(List<Integer> li) {
    int sum = 0;
    for (Integer i : li)
        if (i.intValue() % 2 == 0)
//compiler uses the intValue() method in Integer class to do auto unboxing.
            sum += i.intValue();
        return sum;
}
```

# How do you clone an array? What happens when you clone it?



- Invoking clone() method on an array returns the reference to anew array which contains(references) the same elements as the source array.

- Ex: Code

# Is String a Primitive type in Java?



- NO
- String is a class in Java. It is treated as a object type.

# What does it mean by String is immutable in Java?







- Immutable means unmodifiable unchangeable
- Once string object is created its data or state can't be changed but a new string object is created.
- Every modification to a String object will create a new String object in the Heap memory.
- Immutable objects in Java are thread safe, i.e. Strings are thread safe.
  - String can not be used by two threads simultaneously.

Ex:

Code

# Then Why is it an advantage to have string as immutable?



- Advantage of String being immutable is that it executes as a separate thread.
- Performing string manipulation on strings create new string objects every time.
- To avoid this we use StringBuilder class.

# What are StringBuilder and StringBuffer Objects?







## StringBuffer

- It is mutable, stored on heap.
- Same methods as StringBuilder, and methods are synchronized, i.e. thread safe
- Slower
- String Buffer can be converted to the string by using toString() method.

## StringBuilder

- It is mutable and stored on heap.
- Methods are not synchronized, i.e. StringBuilder is not thread safe.
- StringBuilder is faster than StringBuffer.
- String Builder can be converted to the string by using toString() method.

# Is an Array of chars is same as a String in Java?



- NO.
- Array of chars can be treated as string in C.
- But in Java, Array of chars are mutable, i.e. changeable where as a String object is immutable.
- toCharArray() method in String class can be used to convert a string into a sequence of chars.

Ex: StringBuilder and toCharArray()

Code

# What is the difference between == and equals() method?



- == compares 2 objects memory locations, i.e. it checks whether both the objects are pointing to the same memory locations in the heap or not?
- equals() method is defined in **Object** class.
- equals method is actually meant to compare the contents of 2 objects, and not their location in memory

Ex:

Code

# What is the use of **this** keyword in Java?



- This keyword is used to disambiguate a member variable from a local variable.
- Another use is accessing the instance of the parent class from the nested inner class.
- this() can be used to invoke current class constructor.
- this keyword can be used to invoke current class method (implicitly)

Ex:

Code

# What is garbage collection?



- Garbage collection is a process of de-allocating the memory of objects from the heap. It is used to reclaim the memory for unused objects.
- Java object is garbage collected when it becomes unreachable to the program in which it is used.
- Garbage Collection is automatic in Java. JVM takes care of freeing the memory.

# Can you invoke garbage collection programmatically?





- Yes.

Syntax:

```
Runtime r = Runtime.getRuntime();
r.gc();
```

Note: This is just a suggestion to JVM to run Garbage collection. It doesn't guarantee collection will happen.

To make it simple, Garbage Collection is not in our hands.

# What is **finalize()** in Java? Is it same as the destructor in C++ ?



- Finalise method is invoked by JVM to reclaim the inaccessible memory location.
- JVM invokes finalise() before garbage collection to perform some important cleanup actions.
- Finalise() method can be overridden so as to perform some necessary actions prior to garbage collection.
- It is not same as destructor in C++ as it always destroys objects.

# What is the major drawback in Java that occurs because of poor programming?



- Memory Leaks
- If the program holds a reference to a heap chunk that is not used during the rest of its life memory leak will happen.
- Memory cannot be freed and reused as GC cannot reclaim due to the reference being held by the program.

- Occurs mostly due to programming errors.

# What is the difference between a pointer and reference?



- **Pointer** is an independent variable and can be assigned NEW address values; whereas a **reference**, once assigned, can never refer to any new object until the variable goes out of scope

# What is method overloading?



- Method overloading falls under Polymorphism concept.
- **Method overloading** deals with the notion of having two or more methods(functions) in the same class with the same name but different arguments.
- Can you overload main() method in Java?

# What is Overriding?



- **Method** overriding means having two **methods** with the same arguments, but different implementation. One of them would exist in the Parent class (Base Class) while another will be in the derived class(Child Class).
- A method declared final cannot be overridden.
- A method declared static cannot be overridden but can be re-declared.
- If a method cannot be inherited, then it cannot be overridden.
- Constructors cannot be overridden.

Can we override main() method in Java?

# What is the difference between Array and ArrayList?





Vamsee Krishna Kiran M, Asst.Prof, Dept. of CSE, AVVP

## Array

- Collection of primitive data values
- Stores data as such based on the data type of the array
- If we declare an integer array it accepts only integer

## ArrayList

- Collection of Object types
- Performs boxing and then adds the elements to the collection
- As ArrayList stores as Object type, it can store any type like integer, char, float, string etc.

# What are collections?



- List
- Set
- ArrayList
- HashTable and HashMap
- Iterator
- Dictionary (Key, Value)

# Difference between for and for-each?

- For is a normal control loop.
- Foreach is an iterator that is generally used on collection types.

# What is an exception? What are the blocks used for exception handling?





- Exception is an abnormal condition or an error that is raised in a code sequence during runtime.

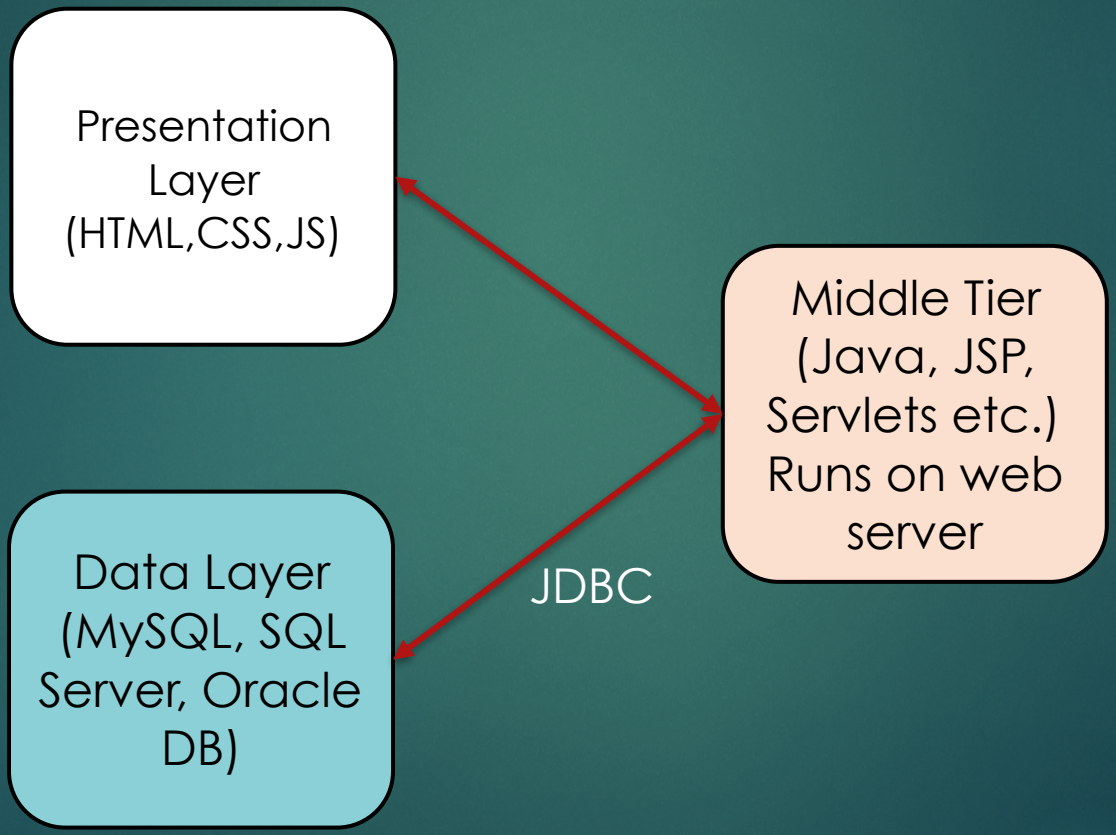
- Try
- Catch
- Finally
- Throw
- Throws

Ex:

Code

# What is three tier architecture?





# Driver Types



- Type-1 JDBC-ODBC Bridge
- Type-2 JDBC Native API - converts calls to native c and c++
- Type-3 JDBC Net Pure Java – Uses network sockets with three tier
- Type-4 100% pure Java-Vendor specific

- Which one to be used?

# What is JDBC?



- JDBC is a JAVA API used to access any kind of tabular data specifically relational databases.
- Works on Mac, Windows, Linux variants.
- Steps involved in accessing DB:
  1. Making connection to DB
  2. Creating SQL or MySQL statements
  3. Executing statements on DB
  4. Viewing or modifying the results.

# JDBC Components



- DriverManager
- Driver (we don't interact directly with this)
- Connection
- Statement
- ResultSet
- SQLException

```
//STEP 1. Import required packages
import java.sql.*;

 Connection conn = null;
  Statement stmt = null;
  try{
     //STEP 2: Register JDBC driver
     Class.forName("com.mysql.jdbc.Driver");

     //STEP 3: Open a connection
     conn = DriverManager.getConnection("jdbc:mysql://localhost/
EMP",USER,PASS);

     //STEP 4: Execute a query
     stmt = conn.createStatement();
     String cmd;
     cmd= "SELECT id, first, last, age FROM Employees";
     ResultSet rs = stmt.executeQuery(sql);
```

```
//STEP 5: Extract data from result set
while(rs.next()){
        //Retrieve by column name
        int id  = rs.getInt("id");
        int age = rs.getInt("age");
        String first = rs.getString("first");
        String last = rs.getString("last");

        //Display values
        System.out.print("ID: " + id);
        System.out.print(", Age: " + age);
        System.out.print(", First: " + first);
        System.out.println(", Last: " + last);
     }
//STEP 6: Clean-up environment
     rs.close();
     stmt.close();
     conn.close();
  }catch(SQLException se){
     //Handle errors for JDBC
     se.printStackTrace();
  }
```

# Statements and Execute methods:



- Statements
  1. Statement
  2. PreparedStatement
  3. CallableStatement
- Execute methods
  1. execute()
  2. executeUpdate()
  3. ExecuteQuery()

# References and Java Sources



1. Complete reference Java Text Book.
2. http://www.tutorialspoint.com/
3. Interview questions from various other Java forums.
4. http://www.indiabix.com/java-programming/questions-and-answers/