OOPS + Java part 1

Class: User defined blueprint from which objects are created. It represents set of properties or methods that are common to all objects of one type.

Class declaration can include:

1. Modifiers
2. Superclass
3. Classname
4. Interface
5. Body

Object: Basic unit of OOP, represents real life entity and we interact by invoking methods.

- States/ Attributes: variables (properties of object)
- Behavior: methods of object
- Identity: unique name of object

When and object of class is created, the class is said to be instantiated.

We can't create objects of abstract class or an interface.

New: it instantiates a class by allocating memory for a new object and returning a reference to that memory. new operator also invokes the class constructor.

If a constructor is not explicity declared the a default constructor is provided by java compiler. This is also called a sclass parent no arg constructor or the object's class constructor. This is because default constructor has super() in it.

Ways to create object of a class:

- using new keyword.
- using class.forName(string className) method: class should be public.

```
Test obj= (Test) Class.forName("package-name.Test").newInstance()
```

- using clone method

```
Test t1= new Test()
Test t2=(Test)t1.clone();
```

Deserialization: during deserialization we recreate the actual java object from bytestream using readObject() method.

Anonymous Objects: objects that are instantiated but not stored in a reference variable.

- used for immediate method calling
- destroyed after method calling
- Ex: AWT Libraries, perform an action on capturing an event (click)

```
btn.setOnAction(new EventHndler()
{public void handle(ActionEvent event)
  { //
    }
  });
//here an anonymous object of event handler class is
//created just for calling handle method
```

Storage of JAVA objects in Memory: All objects are dynamically allocated on heap. When we declare a variable only reference is created. To allocate memory new() is used.

Swap or Exchange objects in Java: We cannot simply create a temp variable and swap them. We use a wrapper class

```
class car
{ int model,num;
Car(int m, int n)
{ this.model=m;
this.num=n;}
void print ( )
{ print(no +" "+model); }

class CarWrapper
{ Car c;
  CarWrapper(Car car)
  { this.c= car;}
```

```
}
public stati void swap( CarWrapper cw1,CarWrapper cw2)
{ Car a= cw1.c;
  cw1.c= cw2.c;
  cw2.c= temp.c; }
public static void main (stirng[] args)
{ Car c1= new Car(101,1);
  Car c2= new Car(202,2);
  Carwrapper cw1= new CarWrapper(c1);
  CarWrapper cw2= new CarWrapper(c2);
  swap(cw1, cw2);
}
//thus we can swap objects even though we do not have access to the
//members of the class.
```

Inheritance

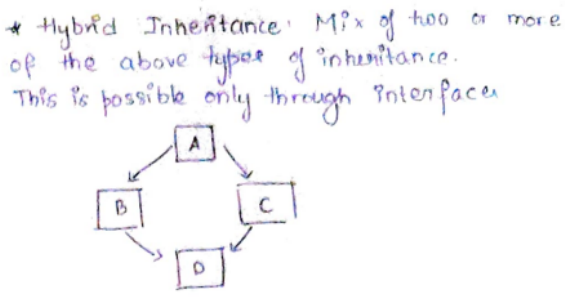
a mechanism in java by which one class is allowed to inherit the features of another class.

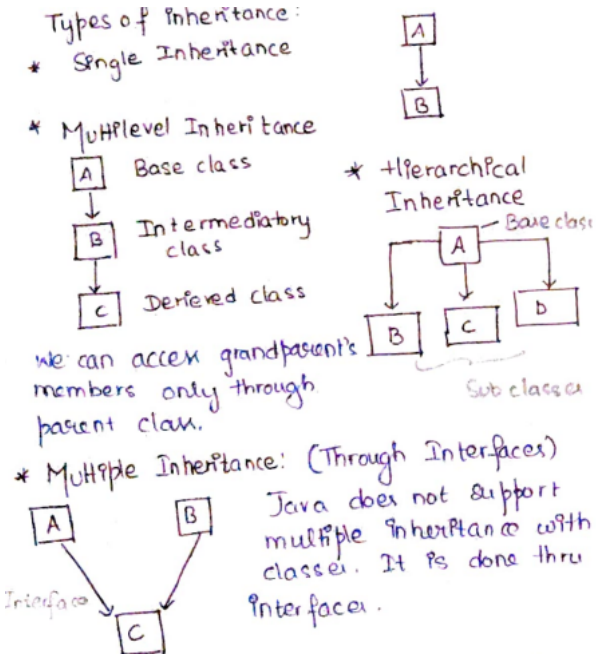
Reusability: reusing methods and fields of existing class.

Keyword for inheritance is extends

Types of inheritance:

- Single inheritance
- Multilevel inheritance
- Hierarchical Inheritance
- Multiple Inheritance
- Hybrid Inheritance






```
interface one { }
interface two { }
interface three extends one, two { }
class child implement three { }
```

Key points on Inheritance

- Absence of explicit superclass, every class is implicitly a subclass of object class.
- A superclass can have number of subclass, but a subclass can have only one superclass.
- Java doesn't support multiple inheritance
- A subclass inherits all member (fields, methods, and nested classes) from its superclass. Constructors are not included in this.
- A private member cannot be accessed by a subclass. It can access public and protected.
- We can write a subclass constructor that invokes the constructor of the superclass, either implicitly or explicitly.

Advantages of Inheritance:

- provides code reusability
- enables data hiding
- run-time polymorphism
- method overridding
- Inheritance of classes with the real time objects makes OOPs more realistic

Encapsulation

Also known as data-hiding

It is a mechanism that binds together code and data it manipulates. it is like a protective shield that prevents the data from being accessed by the code outside the shield.

Encapsulation is achieved by declaring all the variables in the class as private and writing public methods in the class to set and get values of variables.

Advantages of encapsulation:

- Data hiding : hide the inner implementation fro user.
- Increased flexibility as we can make the class as read only ore write only depending on our requirement.
- This improves the re-usability and easy to change code.
- Encapsulated code is easy to test for unit testing

Polymorphism

Having many forms

this allows us to perform a single action in different ways

Allows you to define one interface and have multiple implementations

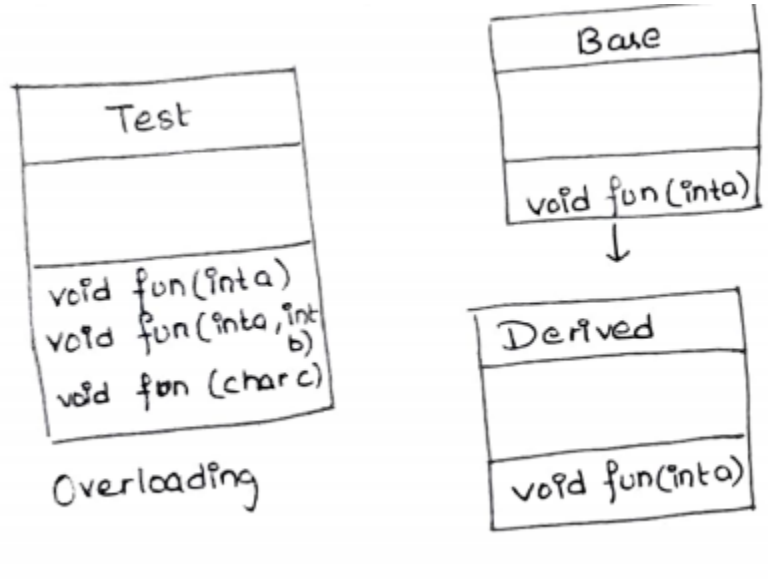
Two types:

- Compile time polymorphism: achieved by function overloading

  also known as static polymorphism

  Functions can be overloaded by changing the number of argument s or/and changing the types of arguments

    - Method overloading
    - Operator overloading = '+' for strings/integers
- Runtime polymorphism: achieved by method overriding. Also known as dynamic method dispatch



Abstraction

Process of identifying only the required characteristics of an object ignoring the irrelevant details. Only essential details are displayed to user. Trivial units are not displayed to user.

- In java, it is achieved by interfaces and abstract classes
- Abstract method does not have implementation, only declaration
- An abstract class may or may not have all abstract methods.
- Method defined as abstract must always be redefined in subclass. Overriding is thus compulsory or make subclass abstract
- Any class that contains abstract method must also be declared as abstract.
- No object of abstract class cannot be instantiated with new operator.
- Abstract class can have parameterized constructors and default constructors is always present.

Encapsulation vs Data Abstraction

- Encapsulation is data hiding while abstraction is detail hiding (implementation hiding)
- Encapsulation groups together data and methods that act upon the data, data abstraction deals with exposing the interface to the user and hiding the details of implementation.

Advantages of Abstractions

- Reducing the complexity of viewing things.
- void code duplication and increase reusability
- Increase security of an application.

Interfaces

Just like a class, but methods in it are by default abstract.

Interface specify what a class must do and not how. Blueprint of the class.

If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must by declared abstract.

Methods are declared with empty body and all fields public, static and final by default. Methods are public and abstract.

A class that implements interface must implement all methods declared in it.

Why use Interface?

- to achieve total abstraction
- for multiple inheritance
- achieve loose coupling
- Implement abstraction

We use interface instead of abstract classes as interface variables are final, public and static whereas abstract classes contain non-final variables.

We can define static methods in interfaces, can be called without an object. Example: interface_name.method_name();

Note: these methods are not inherited.

We can't create instance of interface but we can make reference of it that refers to the object of it's implementing class.

Abstract class vs Interface

- Types of mthods

  Interface: only abstracted method

  Abstract class: both abstract and non- abstract class.

- Final Variables

  Interface: final

  Abstract class: can have non-final variables

- Types of variables:

  Interface: static and final only.

  Abstract class: final/ non-final and static/ non-static

- implementation

  Abstract class can provide implementation of interface while an interface can't provide implementation of the abstract class.

  Multiple implementation: An Interface can extend another interface, while an abstract class can extend another Java class and implement multiple interface.

- Accessibility of data members

  Interface: members are public

  Abstract class: can have members like private, protected etc

Modifiers in Java

Two types of modifiers in Java

- Access modifiers
  - Default: accessible only within the same package
  - Private: accessible only within class

- Protected: Accessible in same package or subclass in different package.
- Public: accessible from everywhere.

- Non- access modifiers

  These are used with classes, methods, variables, constructors etc to provide info about their behavior to JVM

  - Static
  - final
  - abstract
  - synchronized
  - transient
  - volatile
  - native

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non- subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non subclass	No	No	No	Yes

Static vs Dynamic Binding:

Static: can be resolved at compile time by compiler

why binding of static, final and private methods is always static binding?

- Better performance wise

- Such methods cannot be overridden and will always be accessed by object of local class.
- Thus, compiler does not have any difficulty to determine object class.
    - thus no method is overridden for method in superclass

Dynamic Binding: Compiler doesn't decide the method to be called.

Binding would be delayed to runtime.

Important points:

- private, final and static use static binding while all methods by default use dynamic binding.
- static binding use type information for binding while dynamic binding uses objects to resolve binding
- Overloaded methods are resolved using static binding while overriding methods use dynamic binding.

Association

Relation between two separate classes which establishes through their objects.

Association can be one-to-one, one-to-many, many-to-oneetc.

In OOP an object communicates to other object to use functionality and services provided by the object.

Composition and Aggregation are the two forms of asosciation

Aggregation

'Has A" relationship between objects

A unidirectional association, i.e. one way relationship is present.

Both entries can survive individually, independent of each other.

Example: department has students

Compostion

Restricted form of aggregation where two entities are highly dependent on each other.

Represents 'part of' relationship

Both entities are dependent on each other.

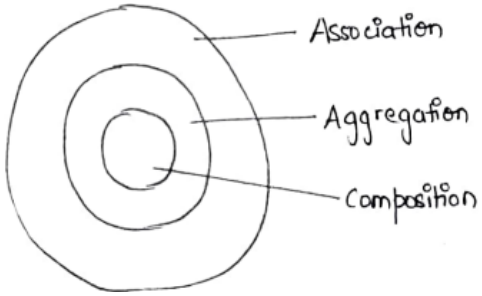
In a composition, a composed object cannot exist without the other entity.

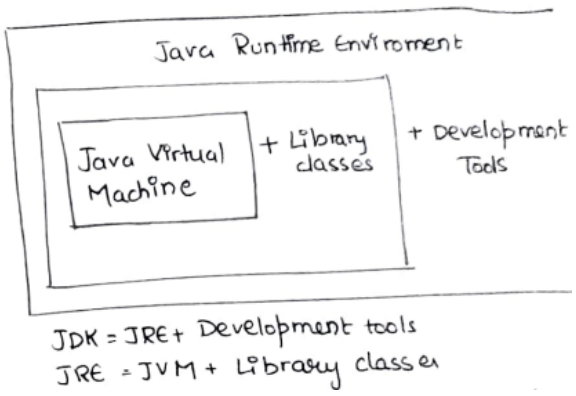
Example: Library and Books

Aggregation vs Composition

Aggregations: child can exist independently of the parent. Composisnition is not like that, they cannot exist independently. Ex: human and heart

Composition is a strong association while aggregation is a weak association.





Difference between JRE, JDK and JVM

Java development Kit: software development environment used for developing Java application. Includes JRE, and interpreter/ loader Java), a compiler, an archiver (jar), a documentation generator: all toots needed in Java Dev

Java Runtime Env: Provides the min requirement for executing a Java application.

Consists of JVM, core classes and supporting files.

Java Virtual Machine: The entity that is responsible for executing the java program line by line. Also known as interpretor

JDK provides an env to develop and execute (run)the java program → development tools → JRE

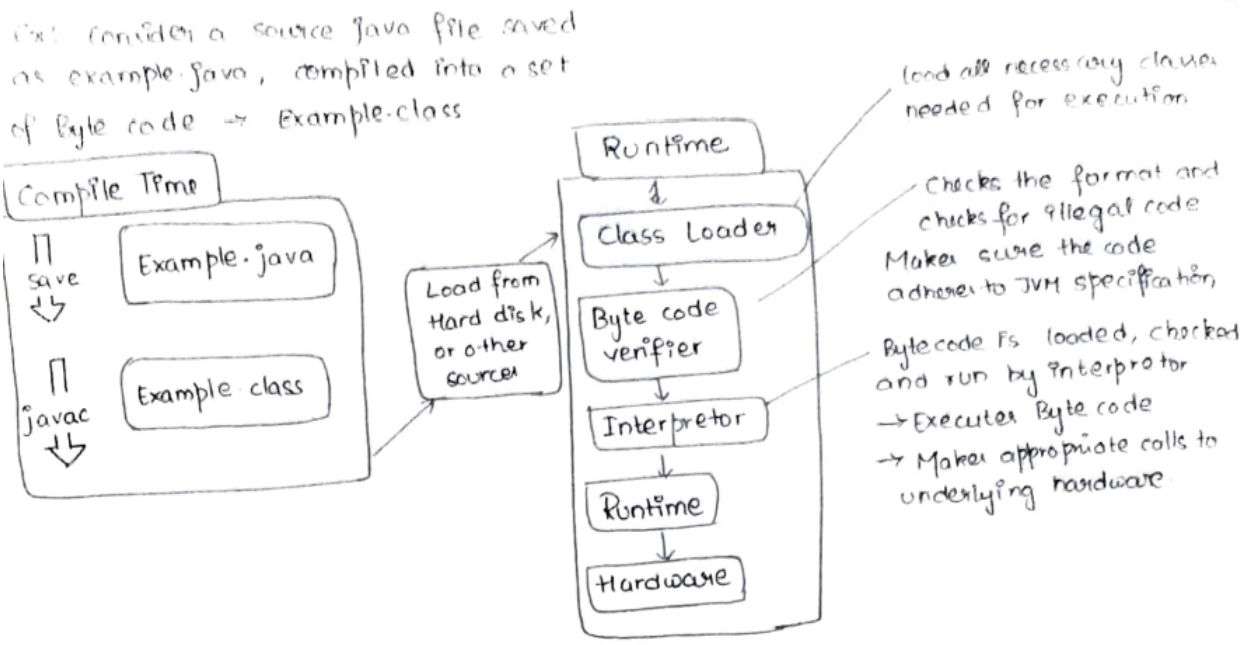
JRE is an installation package which provides environment to only run(not develop) the java program on the machine.

Note: JVM is platform dependent

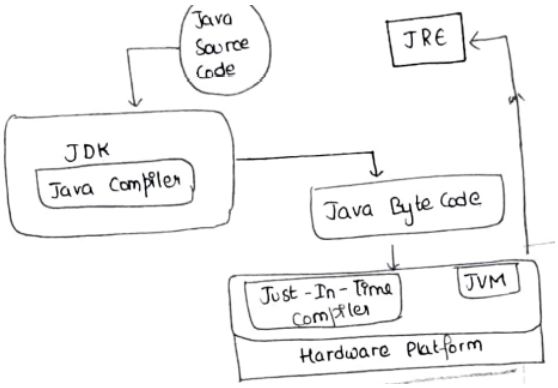
JVM: Provides the env

JRE: required to run code

JDK: required to run code + program



JVM becomes an instance of JRE at runtime of a Java Program.
JVM largely helps in abstraction of inner implementation from the programmer who make use of libraries for from JDK.



String Buffer

Strings represent fixed-length immutable character sequence while string buffer represents growable and writable character sequences.

By default stringBuffer() allows 16 characters but we can specify the length also.

Imp methods:

- Length()
- append()
- insert(int index, String str)
- reverse()
- delete(int start_Index, int end_index)
- deleteCharAt(int index)
- replace(int start, int end, String str) start to end-1 are replaced

String Buffers are safe for use by multiple threads. Methods can also be synchronized.

String Builder

This can be used to create a mutable sequence of characters. It's function is very much similar to that of string Buffer class. It differs in terms of synchronization. String Buffer provides an guarantee of sync where as String Builder does not. String builder is not safe for use by multiple threads, but otherwise it is

recommended that this class be used in preference to string buffer as it will be faster under most implementations.

- append(String str)
- capacity ()
- delete (int start, int end)

Wrapper class in java

we can wrap a primitive value into a wrapper class object.

When we create an object of wrapper class, it contains a field, where we can store a primitive data type.

Why do we need it?

- convert primitive datatypes to objects
- classes in java.util package handle only objects
- Data structures in the collection framework store only objects
- An object is needed to support synchronization in multi-threading

Autoboxing: Automatic conversion of primitive types to the obj of their respective wrapper class.

```
char ch='a'
Character cha= ch
```

Unboxing: Reverse process of autoboxing. Automatically converting an object of wrapper class to its corresponding primitive data type.

```
Character ch='a'
char cha= ch
```