OOPs (CMP211)

Prepared By: Ms.Darshana Makani

Introduction to JAVA:

- JAVA is a object oriented multiprogramming and multi programming system.
- JAVA is developed by Sun Micro System.
- JAVA file must be save with .java extension.
- JAVA is a case sensitive language.

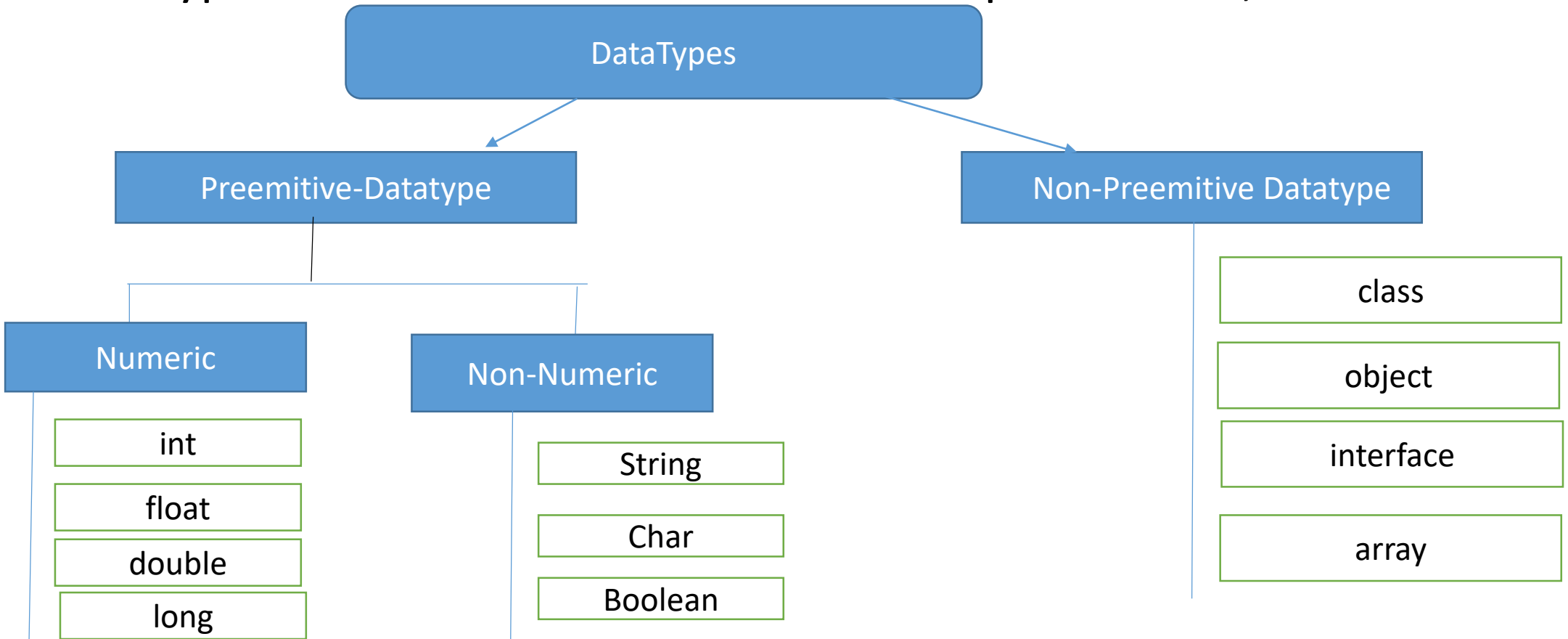
Features of Java:

1. **Simple:** Java is a simple language because its syntax is simple, clean, and easy to understand.
2. **Object-Oriented:** In Java, everything is in the form of the object. It means it has some data and behavior. A program must have at least one class and object.
3. **Robust:** Java makes an effort to check error at run time and compile time. It uses a strong memory management system called garbage collector. Exception handling and garbage collection features make it strong.
4. **Secure:** Java is a secure programming language because it has no explicit pointer and programs runs in the virtual machine. Java contains a security manager that defines the access of Java classes.

5. **Platform-Independent:** Java provides a guarantee that code writes once and run anywhere. This byte code is platform-independent and can be run on any machine.
6. **Portable:** Java Byte code can be carried to any platform. No implementation-dependent features. Everything related to storage is predefined, for example, the size of primitive data types.
7. **High Performance:** Java is an interpreted language. Java enables high performance with the use of the Just-In-Time compiler.
8. **Distributed:** Java also has networking facilities. It is designed for the distributed environment of the internet because it supports TCP/IP protocol. It can run over the internet. EJB and RMI are used to create a distributed system.
9. **Multi-threaded:** Java also supports multi-threading. It means to handle more than one job a time.

DataTypes:

- Datatype is defining a type of declaring variable and also describe which type of value stored in variable.
- Datatype are converted into two different parts as like,

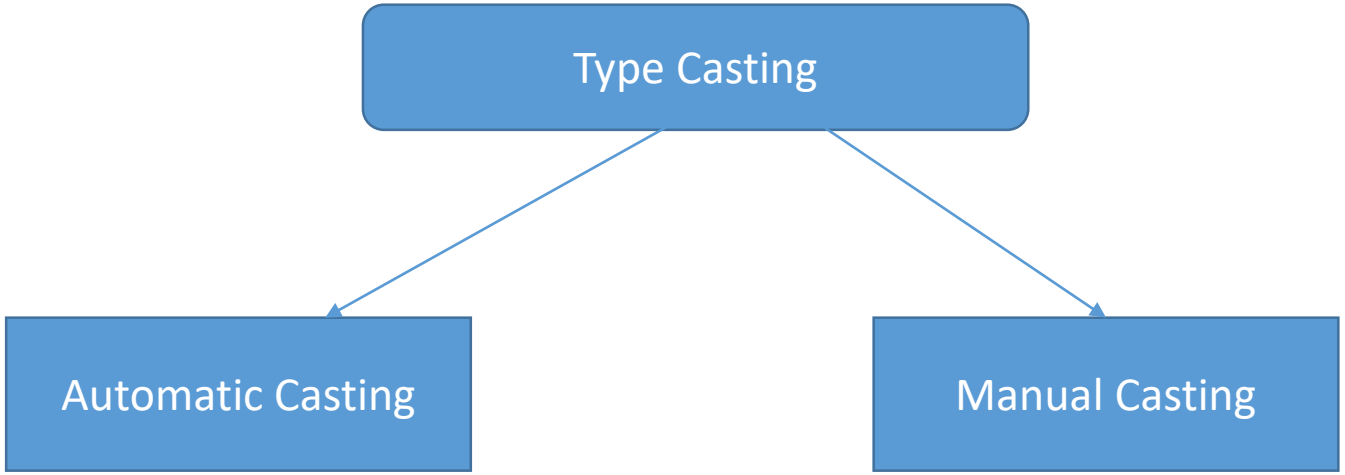


- Example:

```
Class datatype1
{
    Int a=101;
    Char c='c';
    String sname="abc";
    Boolean answer=true;
    Float per=89.99;
    }
```

❖ Type-Casting:

- Type casting is a one process or method like converting one datatype data into another datatype.
- Type casting process are divided into two types as like,



1.Automatic casting:

- Converting a smaller size data type into a higher one is called **automatic** type casting.
- It is also known as **implicit conversion** .
- It is done automatically. It is safe because there is no chance to lose data.
- It takes place when:
  - Both data types must be compatible with each other.
  - The target type must be larger than the source type.
- For Example:

```
Int a=10;
Float b=a;
```

2.Manual casting:

- Converting a higher data type into a smaller size of data is called **Manual** type casting.
- It is also known as **explicit conversion**.
- It is done manually by the programmer.
- If we do not perform casting then the compiler reports a compile-time error.
- Syntax:

  Datatype variable_name=(convert_type)variable_name;

- Example:

  Float a=89.45;

  Int b=(int) a;

❖ Command-line Argument:-

- **command-line argument** is an argument i.e. passed at the time of running the Java program.
- pass the arguments as space-separated values.
- the command line arguments passed from the console can be received in the Java program and they can be used as input.
- The users can pass the arguments during the execution bypassing the command-line arguments inside the main() method.
- Also pass both strings and primitive data types(int, double, float, char, etc) as command-line arguments.

- When command-line arguments are supplied to JVM, JVM wraps these and supplies them to args[].
- It can be confirmed that they are wrapped up in an args array by checking the length of args using args.length.
- first command-line argument at args[0], the second at args[1], the third at args[2], and so on.
- **For Example:**

```
public class com_arg {
    public static void main(String args[]) {
        System.out.println(“Nuvrachana University");
        int c = args.length;
        System.out.println("length of arguments=" + c);}}
```

- **Compile time:-** javac com_arg.java
- **Run time:-** java com_arg 10 20 30

❖ Garbage Collection:-

- The garbage collector finds these unused objects and deletes them to free up memory.
- The main objective of Garbage Collector is to free up memory by destroying **unreachable objects**.
- Java garbage collection is an automatic process.
- Automatic garbage collection is the process of looking at memory, identifying which objects are in use and which are not, and deleting the unused objects.
- In short garbage collection is process for destroying unused object.
- The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing.
- The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

- Following three condition define unreferenced object,
  1. By nulling the reference
  2. By assigning a reference to another
  3. By anonymous object etc.

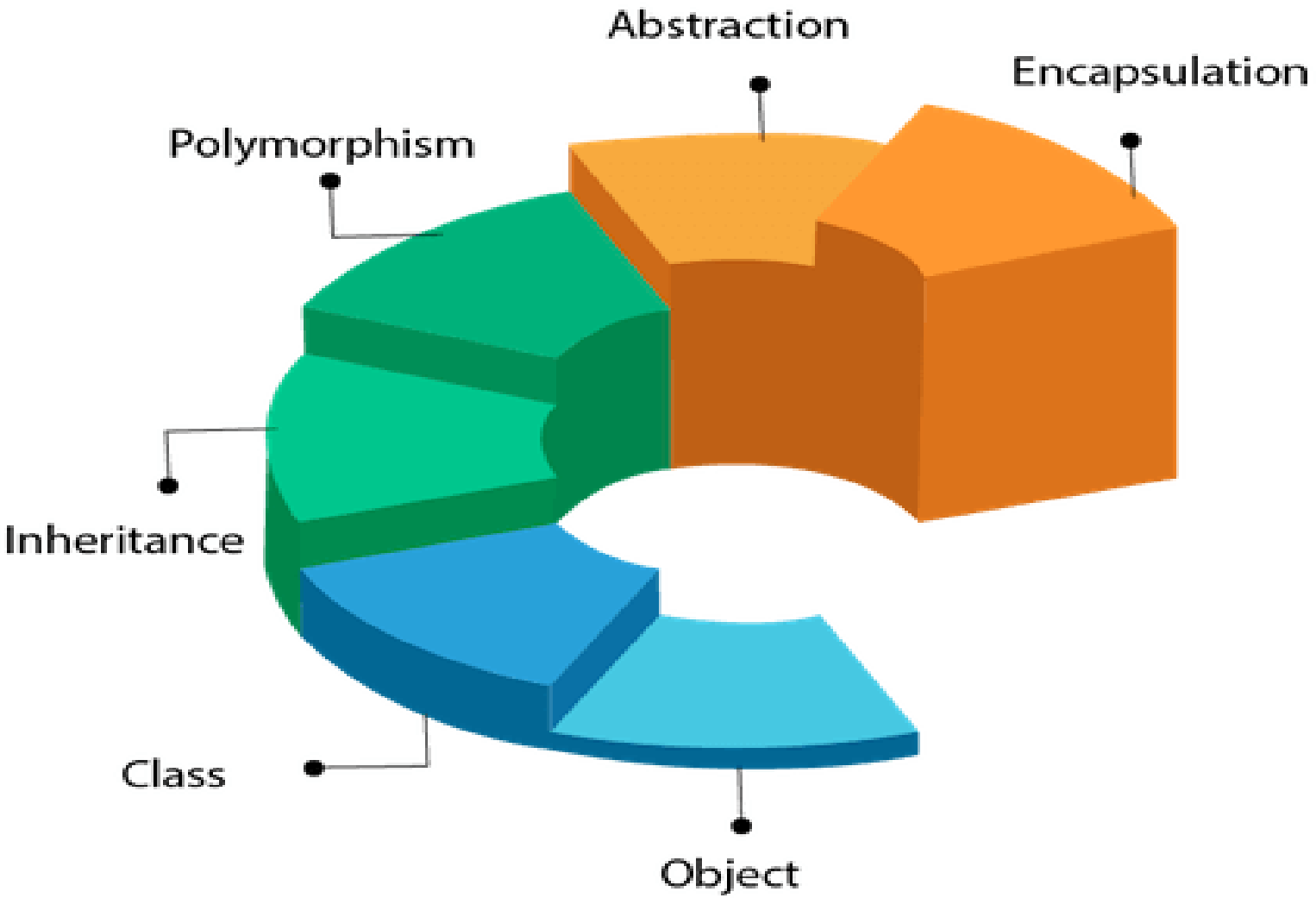
- **For example:-**

```
Class garbage_collection
{
        public static void main(String args[])
{
                        //nulling reference
                        Employee e=new Employee();
                        e=null;
                        //assign reference to another
                        Employee e1=new Employee();
                        Employee e2=new Employee();
                        e1=e2;
                        //anonymous object
                        new Employee();
}
}
```

OOPs Overview:

- OOPs means Object Oriented Programming.
- OOPs are used for defining how to apply programming logic for developing real World applications.
- OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.
- OOPs define real world entities like,

OOPs (Object-Oriented Programming System)



❖ Class & Objects:-

- A class is a group of objects which have common properties.
- It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.
- Object also referred as a real world entity.
- So we also called an entity which has state and behavior is known as object.
- **Syntax of class:**

```
class <class_name>
{
    return-type methodname();
    datatype variableName;
}
```

- Object is created by **new** operator which can be allocate memory space for specified class.
- Syntax of object:
  - Classname object_name=new classname();

- Every properties(variable and method) of class are call by using object.
- Syntax
  - Object_name.variable_name=value;
  - Object_name.methodname();

- **For example:**

```
Class employee
{
        Int eid=101
        Void display()
        {
                System.out.println(“employee id=“+eid);
        }
}
Class  abc
{
                public static void main(String args[])
                {
                        employee e1=new employee();
                        e1.eid=101;
                        e1.display();
                }
}
```

❖ Constructor:-

- A constructor is a **code of block** which is similar look like same as method.
- When we create **method name same as** like **class name** so this created method known as a constructor.
- Constructor are **automatic calling** when we **create object of class**.
- So, constructor **can not be call by object of class**.
- Constructor does **not have any return type**.
- Constructor allocate a memory when we calling object of class.
- A Java constructor **cannot be abstract, static, final, and synchronized**
- Constructor are defining in three ways,

- Types of constructor:
  1. Default constructor
  2. Parameterized constructor
  3. Constructor overloading

1. Default constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- **Syntax of default constructor:**

  <class_name>(){}
- **Example:**

```
Class emp
{
    emp()
    {
        System.out.println(“this is default constructor”);
        }
    }
}
```

2. Parameterized Constructor

- A constructor which has a specific number of parameters is called a parameterized constructor.
- The parameterized constructor is used to provide different values to different variable.
- However, you can provide the same values also.

- **Example:-**

```
Class Student
{
    Int rno;
    String name;
    Student(int r, String n)   //parameterized constructor
    {
        rno=r;
        name=n;
    }
}
```

- When we create object ,we also give some value at the object creation time so this values are passes to parameter of constructor.

```
Student s1=new Student(101,”abc”);
```

3. Constructor Overloading:

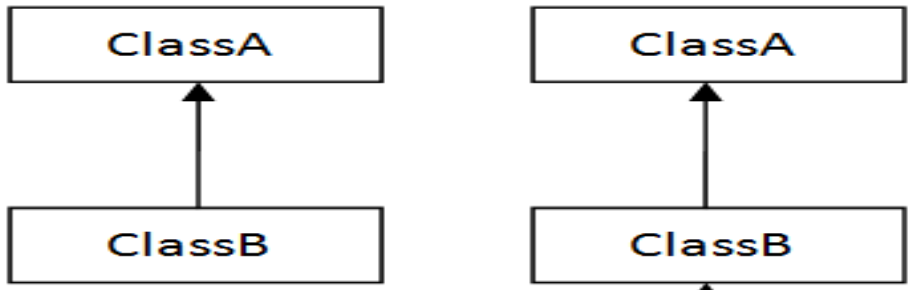
- When we create two or more different type of constructor inside a class so this process called constructor overloading.
- In short, constructor overloading means combination of default and parameterized constructor.
- Here each constructor performs a different task.
- When we create object of class that time first default constructor called then after different parameterized constructor called.

❖ Inheritance:-

- Inheritance means create a new class from existing class.
- Using Inheritance,new generated class access all properties(variable & methods) of existing class.
- Inheritance define IS-A relationship(parent-child relationship) of class.
- Existing class known as parent class/base class/super class.
- New generated class known as child class/derived class/sub class.
- Inheritance are creating by “extends” keyword.
- Using inheritance, parent class properties and child class properties always call by child class objects.
- Syntax:

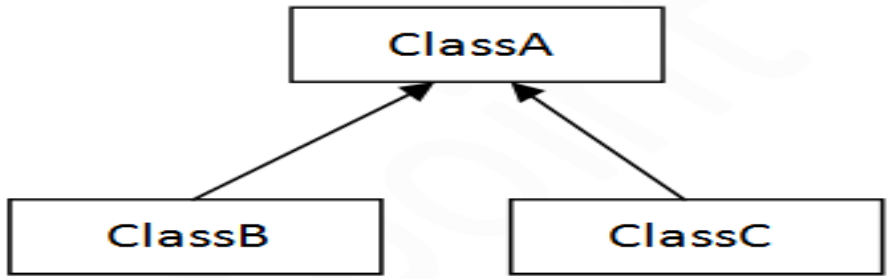
```
class Childclass-name extends Parentclass-name
{
  //methods and  variables;
}
```

- **Types of inheritance:-**

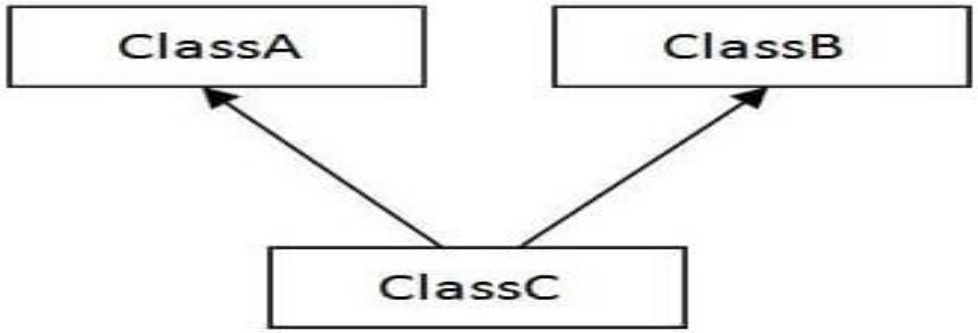


1) Single

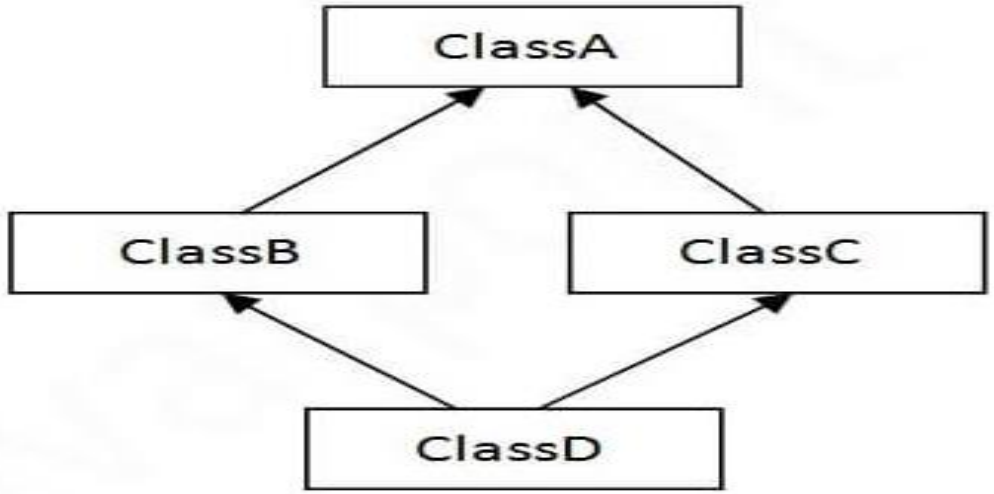
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

- Java only support following 3 type of inheritance like,
    1. **Single Inheritance**
    2. **Multi-level Inheritance**
    3. **Hierarchical Inheritance**
- But java does not support multiple and hybrid inheritance so this concept are solved by interface.

1. Single Inheritance:
    - A class generate only single class this type of inheritance known as single inheritance.
    - Only one parent class and only one child class this concept call single inheritance.
    - (Note: example you have to define)

2. Multilevel Inheritance:

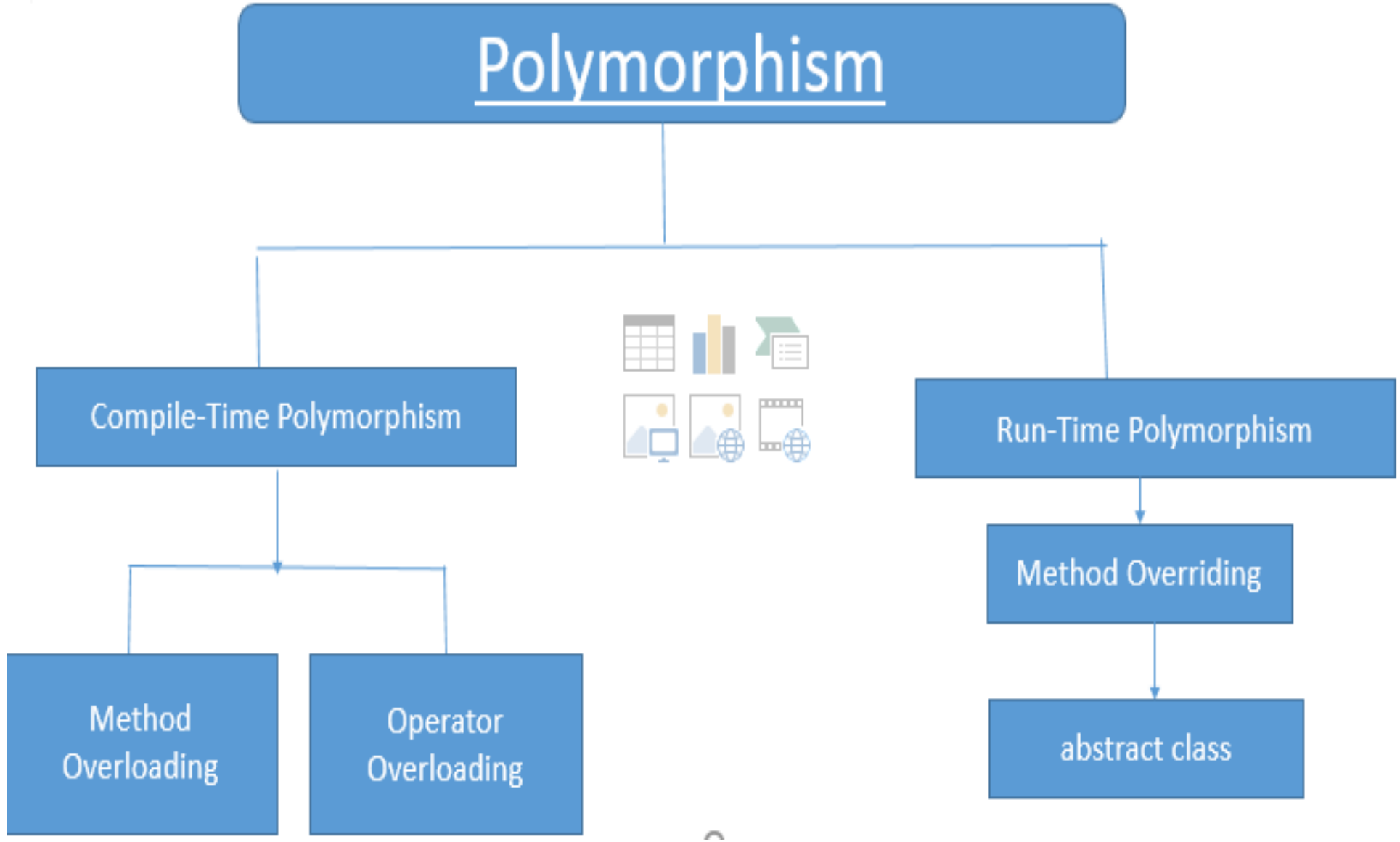
- Here each and every level only one class generated.
- Means that, Only one parent generate one child class and this child class again generate most child class this process known as multi level inheritance.
- Also chain of inheritance is known as *multilevel inheritance*.
- (Note: example you have to define)

3. Hierarchical Inheritance:

- When two or more classes inherits only a single class, it is known as *hierarchical inheritance*.
- (Note: example you have to define)

❖ Polymorphism:-

- Polymorphism is a Greek language word.
- It is a combination of poly and morphism word where poly means ‘many’ and morphism means ‘format/structure’.
- Any object which have multiple format that define by polymorphism.
- Compile time polymorphism define overloading concept.
- Run time polymorphism define overriding concept.


Polymorphism
Compile-Time Polymorphism
Run-Time Polymorphism
Method Overloading
Operator Overloading
Method Overriding
abstract class

❖ Method Overloading:

- Method overloading known as Compile time polymorphism .
- A class which have two or more methods and also all method name are same but return type is different so technically this concept achieved by method overloading.
- In short, method overloading means method name are same but behavior are different.
- Example:

```
Class calculate
{
    Int sum(int a, int b)
    { return a+b; }
    Float sum(float x, float y)
    { return x+y; }
}
```

- There are two ways to overload the method in java
    1. By changing number of arguments
    2. By changing the data type

❖ Abstraction:-

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.
- **Rules for Java Method Overriding**
  1. The method must have the same name as in the parent class
  2. The method must have the same parameter as in the parent class.
  3. There must be an IS-A relationship (inheritance).
- When method overriding are generated that time actual process output is running but not display
- So this problem are solved by abstract class.

- A class which is declared with the abstract keyword is known as an abstract class.
- It can have abstract and non-abstract methods (method with the body).
- Abstraction means “know about what to do but don’t know about how to do”.
- Abstraction is a process of hiding some implementation details and showing only functionality to the user.
- A class which is declared as abstract is known as an **abstract class**.
- It can have abstract and non-abstract methods. It needs to be extended and its method implemented.
- Object of abstract class are not created.

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It can have constructors and static methods also.
- Abstract class only define abstract method but not describe.it describe its child class.
- Here inheritance must be required and abstract class must be declared at parent level.
- Syntax:

```
Abstract class_name
{
    Abstract method_name();
    ……..
}
```

- Example:

```
abstract class Shape
{
        abstract void draw();
}
```

❖ Super Keyword:-

- When we might require parent class variable uses with its same value inside child class that time we can use super keyword.
- The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.
- super can be used to refer immediate parent class instance variable.
- Super() always define as a first statement inside child class constructor.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

- Example:

```
Class  box extends mybox
{
        Int depth;
        Box(int l,int w,int d)
        {
                Super(l,w);
                Depth=d;
        }
}
```

❖ Access Modifier:-

- There are four types of Java access modifiers:
- **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

Access Modifier:-

Modifiers	Inside only class	Outside a class in same file	Access inside/outside files
Private	yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

❖ Encapsulation:-

- Encapsulation means to hide some information/data from the another file that time we can use encapsulation.
- It can wrapping a code by encapsulation.
- The **Java Bean** class is the example of a fully encapsulated class.
- For encapsulation we must declare a data member as a “private”.
- We must create setter and getter method for each and every field.
- Using encapsulation we easily maintain and tested a code.
- Data are controlling by encapsulation.

- For example:

```
class  emp
{

    private int eid;

    public void setEid(int newId)
    {
        this.eid=newId;
    }
    int getEid()
    {
        return eid;
    }

}
```