Java - OOPS

Session Objective

- Class & Object
- Encapsulation
- Inheritance
- Polymorphism
  - Overloading
  - Overriding
- Constructor
- Abstraction
  - Abstract class
  - Interface

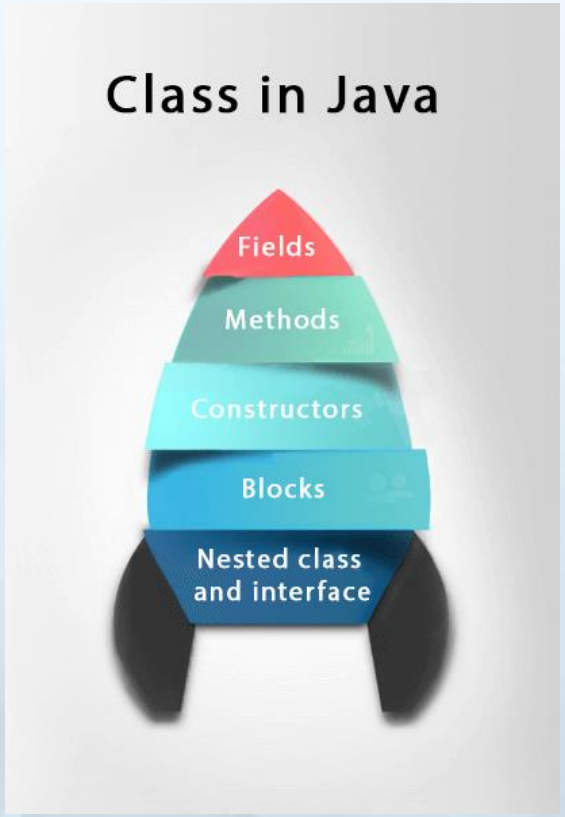
Class & Object

Gamification - OOPs

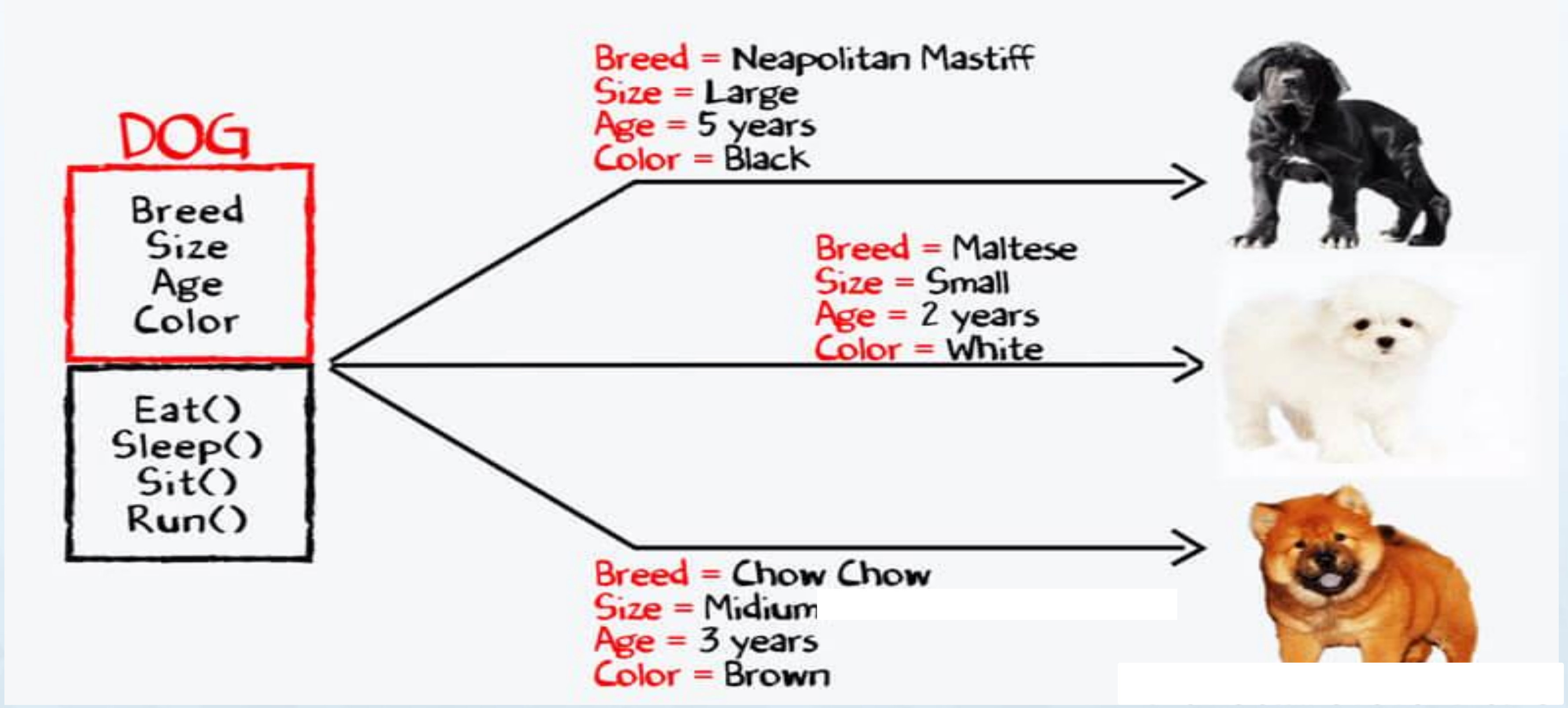
OOPS
Gamification

Class & Object -Contd…

- A class is a template or blueprint from which objects are created.
- Object is the instance of a class.
- The object is an entity which has state and behavior



Class & Object -Contd…



Standards for Class & Method

Class: Usually class name should be noun starting with uppercase letter.

- If it contains multiple word than every inner word should start with uppercase.
  - Eg: EmployeeDetails, StringBuffer

Methods: Usually method name should either be verb or verb noun combination starting with lower letter.

- If it contains multiple word then every inner word should start with uppercase.
  - Eg: print(), sleep(), setSalary()

Class & Object

```
//Class creation
public class Employee{
   private static int empSalary=30000;
   public float empDetails(float bonus){
     float totalAmt;
     totalAmt=empSalary+bonus;
     return totalAmt;
   }
 public static void main(String[] args) {
  //Object declaration
  Employee empObj=new Employee();
  float tsal=empObj.empDetails(5000);
  System.out.println("Total sal:"+tsal);
 }
}
```

Coding Standards

Class Comments

- Every class should be preceded with a descriptive comment using the "JavaDoc" notational convention.
- The comment should describe its purpose of the class.
- Class names start with an upper case letter.

```
/**
 * Stores the first, middle, and last names for a president.
 */
class President {
   //code...
}
```

Coding Standards

Method Comments

- Every method definition should be preceded with a descriptive comment using the "Javadoc" notational convention.
- The comment should include a description of the method, the name and description of each parameter, a description of the return value, and the name and description of any exceptions thrown within the method using Javadoc keywords and formatting.

Note:

@param, followed by the name of the parameter, followed by a description of the parameter

Coding Standards

```
/**
 * Prints a word, prints a number, and returns integer 1
 *
 * @param word any string of Class String
 * @param number an integer of any value
 * @return the integer 1
 */
public static int method1(String word, Integer number) {
  //code...
  return 1;
  }
```

Coding Standards

Class, Package, and Method Naming and Capitalization

- Classes begin with a capital letter.
- Packages are all lower case.
- Methods begin with a lower case letter.
- Multi-word identifiers are internally capitalized in methods (CamelCase).

```
package foo.bar.baz;
public class MeanStandardDeviation{
private Vector getNewVector(Vector oldVector) {}}
```

Encapsulation

Encapsulation Contd…

- Encapsulation in Java is a process of wrapping code and data together into a single unit
- **Example:** Java Bean class is the example of a fully encapsulated class, a capsule which is mixed of several medicines.
- Encapsulation in java is achieved using private, protected and public keywords.

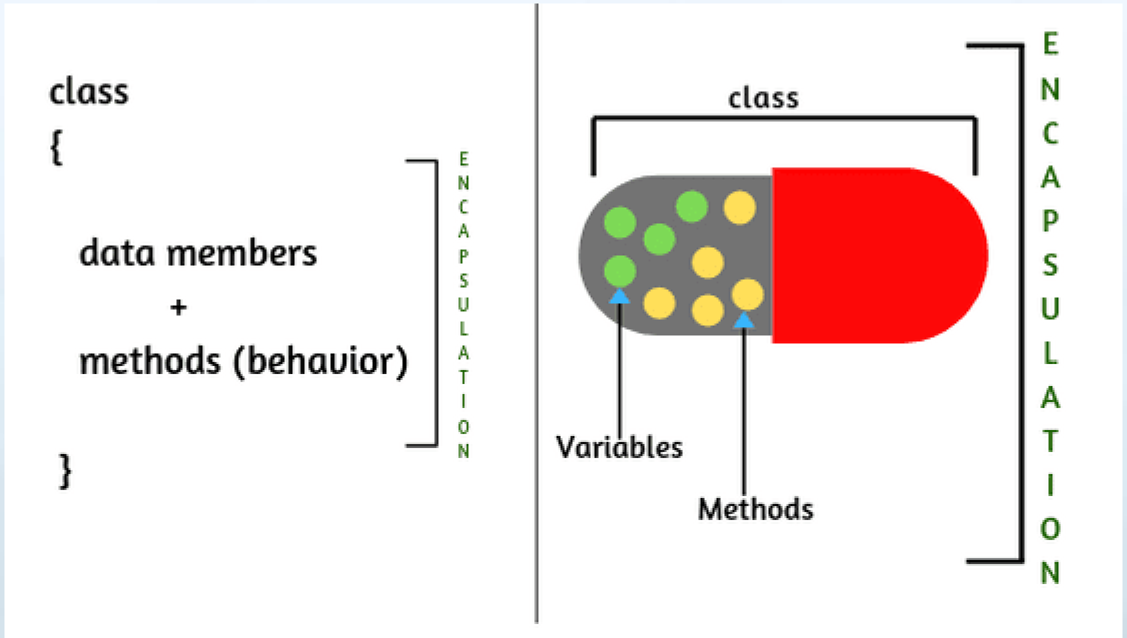


Fig: Encapsulation

Encapsulation Contd…

- In encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
- Encapsulation can be achieved by declaring all the variables in the class as **private** and writing **public methods** in the class to set and get the values of variables.

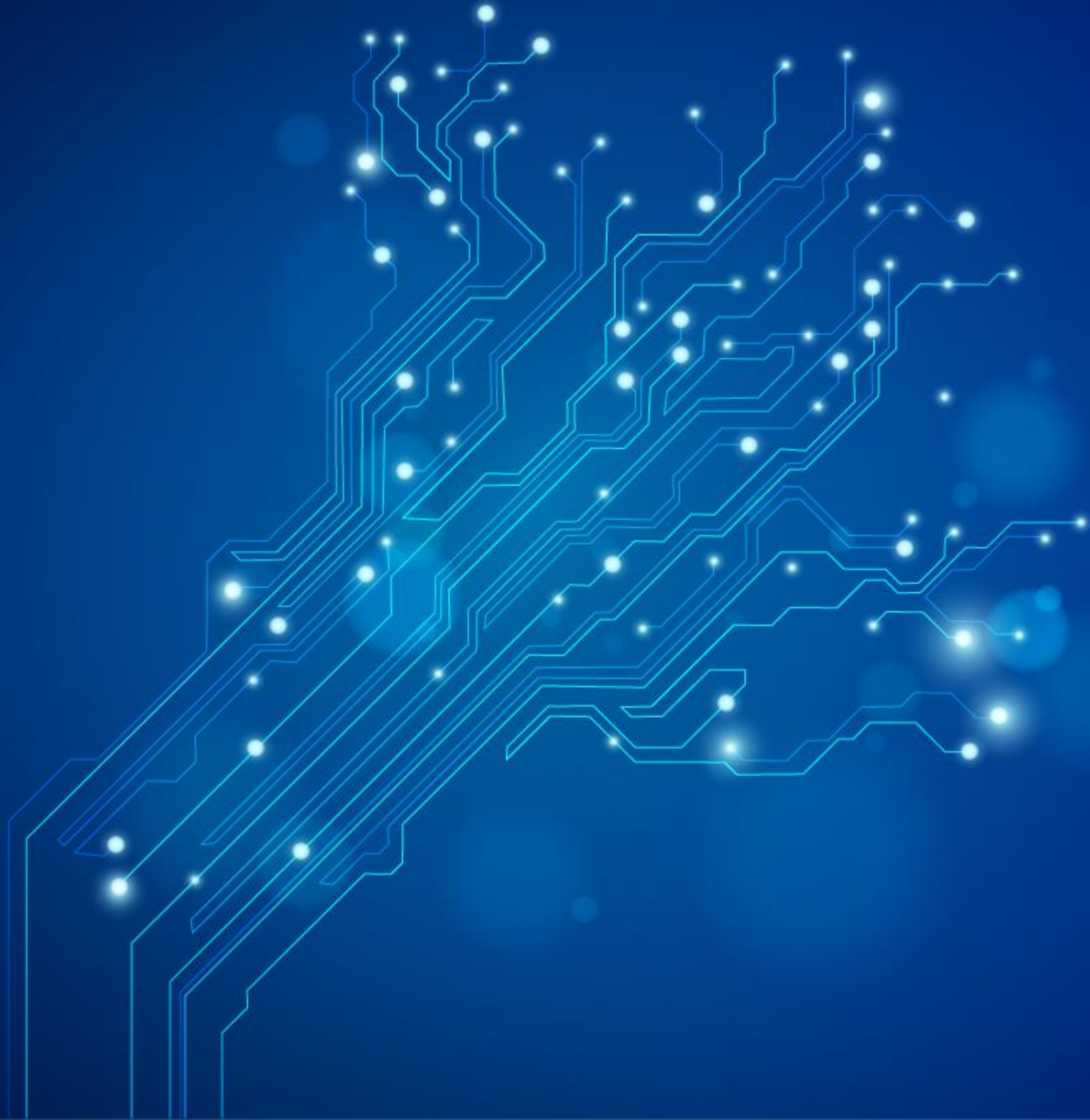
Encapsulation Contd…

```
class Customer{
    private int cid;
    private String cname;
    public Customer(int cid,String name){
        this.cid=cid;
        this.cname=name;
    }
    public void setCid(int cid){
        this.cid=cid;
    }
    public int getCid(){
        return cid;
    }
    public void setName(String cname){
        this.cname=cname;
    }
    public String getName(){
        return cname;
    }
}
```

Encapsulation

Advantages of Encapsulation:

- Data Hiding
- Increased Flexibility
- Reusability
- Testing code is easy



Inheritance

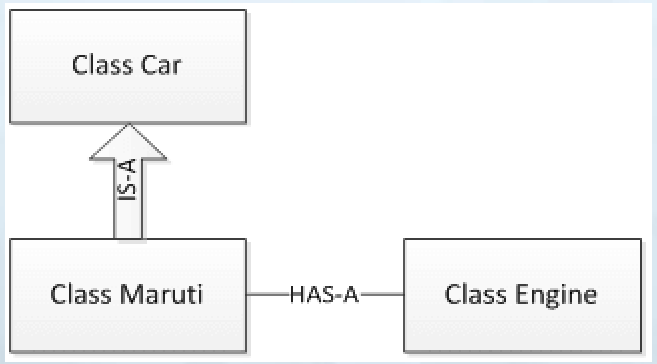
Inheritance Contd…

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.
- For example, Apple is a Fruit, Car is a Vehicle etc. Inheritance is uni-directional.
- **Reusability:** Inheritance supports the concept of “reusability”, i.e. when a new class is created and there is already a class that includes some of the code that we want, we can derive our new class from the existing class.

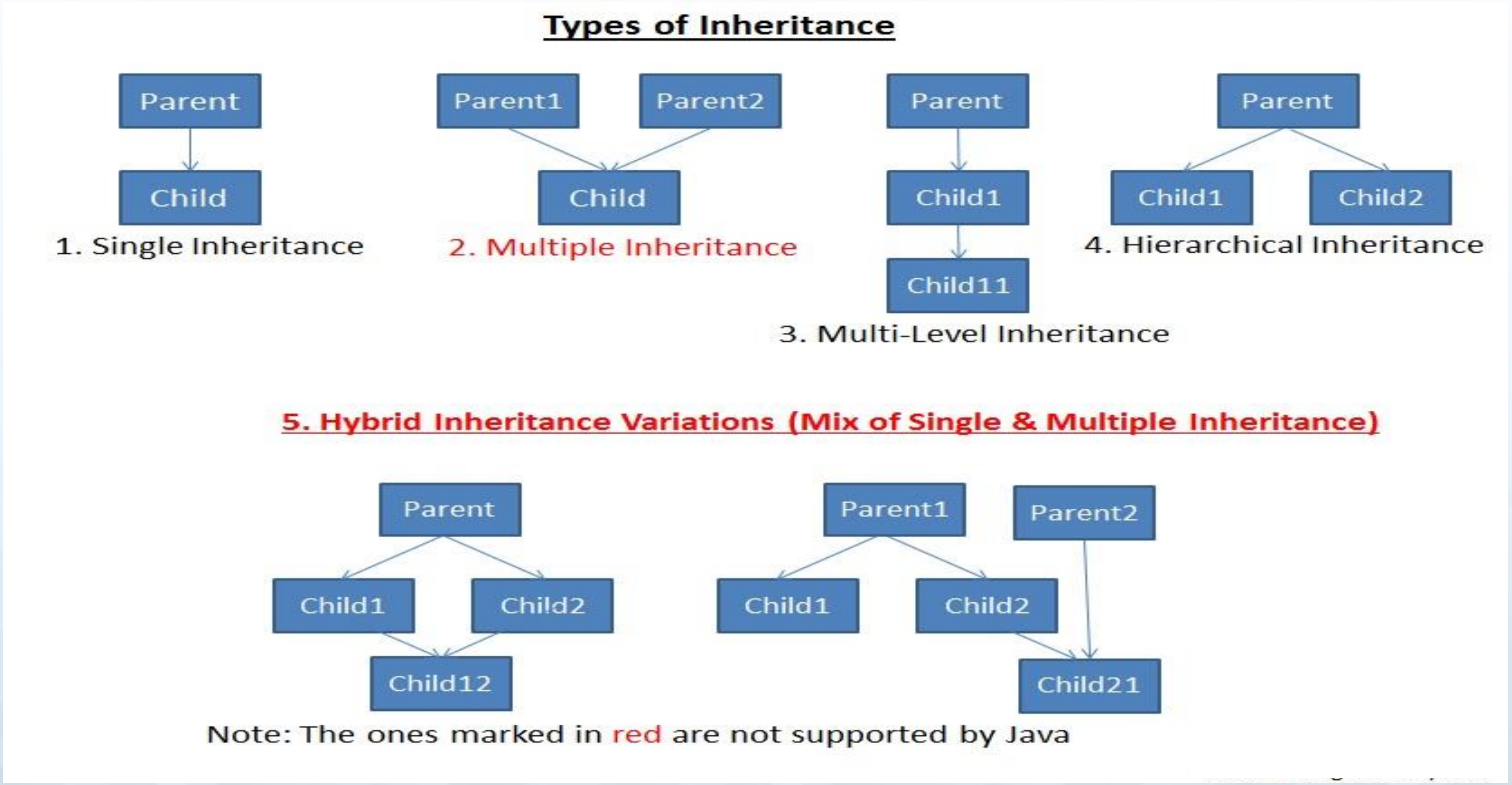
Inheritance Contd…

HAS-A Relationship:

- ✓ Composition(HAS-A) is instance variables that are referenced to other objects.
- ✓ Has-A relationship implies that an example of one class has a reference to an occasion of another class or another occurrence of a similar class.
- ✓ Example: Maruti has Engine.
- ✓ Employee has an object of Address, address object contains its own information such as city, state, country etc. Employee HAS-A address.



Inheritance Contd…



Inheritance Contd…

```
class Training{
    int tid=12;
    String tname="Java";
    public void trgDetails(){
        System.out.println("Training id:"+tid);
        System.out.println("Training name:"+tname);
    }
}
class ADM extends Training{
    String emptype="G3";
    public static void main(String[] args) {
      ADM admObj=new ADM();
      admObj.trgDetails();
      System.out.println("Type of Employee:"+admObj.emptype);
    }
}
```

Think & Answer

Why multiple inheritance is not supported in java?

Inheritance

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
- Example:

  Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Polymorphism

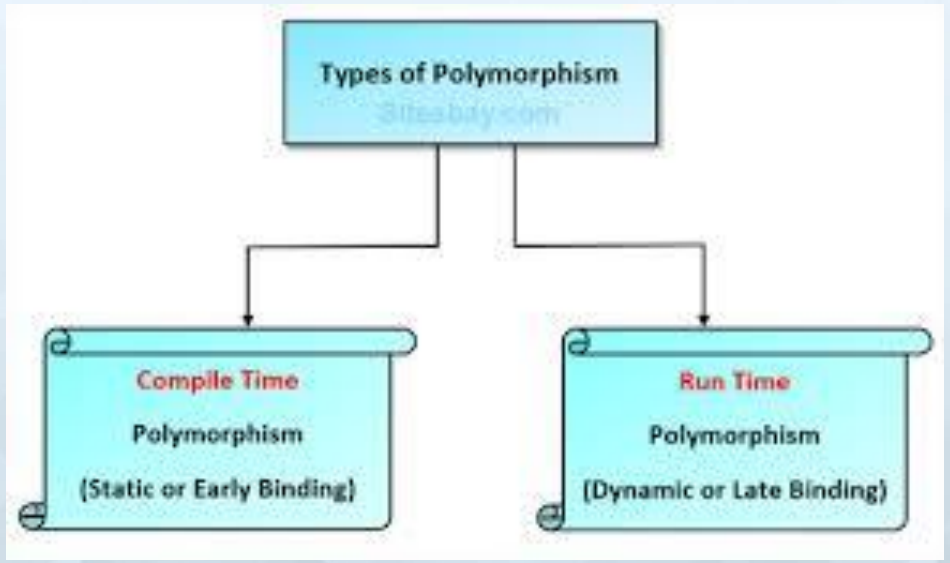
Polymorphism Contd…

- Polymorphism is one of the OOPs feature that allows us to perform a single action in different ways
- The word "poly" means many and "morphs" means forms

A Person
Polymorphism

Polymorphism Contd…

- **Compile time polymorphism:** This type of polymorphism is achieved by function overloading.
- **Runtime polymorphism:** It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.



Overloading Contd…

- Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both.

Overloading

```
class HexaTraining{
    public void angularTrg(String intTr){
        System.out.println("Internal trainer:"+intTr);
    }
    public void angularTrg(String exTr, int cost){
        System.out.println("External trainer:"+exTr+"    "+"Cost:"+cost);
    }
    public void angularTrg(String exTr, int cost,String venue){
        System.out.println("External trainer:"+exTr+"    "+"Cost:"+cost+"    "+"Venue:"+venue);
    }
    public static void main(String[] args) {
        HexaTraining hexa=new HexaTraining();
        hexa.angularTrg("Vimala");
        hexa.angularTrg("Thomas",40000);
        hexa.angularTrg("Kavitha",70000,"Chennai");
    }
}
```

Think & Answer

Can we overload static methods?

Can we overload main() in Java?

Can we overload methods that differ only by static keyword?

Think & Answer

1. The answer is ‘**Yes**’.

   We can have two ore more static methods with same name, but differences in input parameters.

2. We **cannot** overload two methods in Java if they differ only by static keyword
3. Like other static methods, we **can** overload main() in Java.

Overriding Contd…

- When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass override the method in the super-class.

Overriding Contd…

```
class FTP{
   public  void trainingDetails(){
    String trg="Java,SQL,Angular";
        System.out.println("FTP training tec:"+trg);
    }
}
class Competency extends FTP {
    String comTrg="Block chain, Big data";
    public   void trainingDetails(){
        String comTrg="Block chain, Big data";
       // super.trainingDetails();
        System.out.println("Competency training tec:"+comTrg);
    }
public static void main(String[] args) {
    Competency com=new Competency();
    com.trainingDetails();
}
}
```

Overriding Contd…

Overriding and Access-Modifiers :

- ✓ The access modifier for an overriding method can allow more, but not less, access than the overridden method.
- ✓ For example, a protected instance method in the super-class can be made public, but not private, in the subclass.

Final methods can not be overridden :

- ✓ If a method does not require to be overridden, declare it as final.

Static methods can not be overridden(Method Overriding vs Method Hiding) :

- ✓ When a static method is defined with same signature as a static method in base class, it is known as method hiding.

Overriding Contd…

Overriding and constructor :

- ✓ Constructor can not be overriden, as parent and child class can never have constructor with same name(Constructor name must always be same as Class name).

Invoking overridden method from sub-class :

- ✓ Parent class method can be called in overriding method using **super** keyword

Constructor

Constructors Contd…

- A constructor is called when an instance of the class is created.
- At the time of calling constructor, memory for the object is allocated
- Constructors are used to initialize the object's state.
- Each time an object is created using new() keyword at least one constructor is invoked to assign initial values to the data members of the same class.

```
class Finance {
    public Finance(int fid){
            System.out.println("Finanace id:"+fid);
    }
}
```

Constructors Contd…

Rules for Constructor:

- Constructor(s) of a class must has same name as the class name in which it resides.
- A constructor in Java can not be **abstract, final, static and Synchronized**.
- Access modifiers can be used in constructor declaration to control its accessibility.

Types of constructor

No-argument constructor:

- A constructor that has no parameter is known as **default constructor**. If a constructor is not defined in a class, then compiler creates default constructor(with no arguments) for that class

Parameterized Constructor:

- A constructor that has parameters is known as parameterized constructor. Parameterized constructor is used to initialize fields of the class with own values.

Note: If we write a constructor with arguments or no-arguments then the compiler does not create a default constructor.

Constructor Contd…

```
class Hr{
 public Hr(int hrId){
        System.out.println("HR ID:"+hrId);
    }
    public Hr(int hrId,String hrName){
        this(88888);
        System.out.println("HR id:"+hrId+"   "+"HR 
name:"+hrName);
    }
    public Hr(String hrApproval){
        System.out.println("HR Approval:"+hrApproval);
    }
}
class Finance extends Hr{
        public Finance(int fid){
        super("Accepted");
        System.out.println("Finanace id:"+fid);
    }
public  Finance(int fid,String travelPlace){
        super(88,"Hendry");
        System.out.println("Finanace id:"+fid+"
"+"Travel Place:"+travelPlace);
    }
    public static void main(String[] args) {
        Finance fin=new Finance(22);
        Finance fin1=new Finance(444,"America");
    }
}
```

Constructor

Constructor Overloading

- Constructors can be overloaded for creating objects in different ways.
- Compiler differentiates constructors on the basis of numbers of parameters, types of the parameters and order of the parameters.

Think & Answer

Does constructor return any value?

Can constructor perform other tasks instead of initialization?

Can constructor be inherited ?

Answer

1. Yes, it is the current class instance
2. Yes, like object creation, starting a thread, calling a method, etc
3. No, constructor can not be inherited

Constructors are special and have same name as class name. So if constructors were inherited in child class then child class would contain a parent class constructor which is against the constraint that constructor should have same name as class name.

Abstraction

Abstraction

- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- **Example:** Sending SMS, where we type the text and send the message. We don't know the internal processing about the message delivery.
- Ways to achieve Abstraction
  - There are two ways to achieve abstraction in java
    - Abstract class
    - Interface

Abstract class

Abstract Class Contd…

If a class has at least one pure virtual function, then the class becomes abstract class.

```
abstract class Mobile{
static final String internalMemorySize="8GB";
public void calling();
public void texting();
public void fm(){
System.out.println(“98.3 fm”);
}
}
```

Abstract Class Contd…

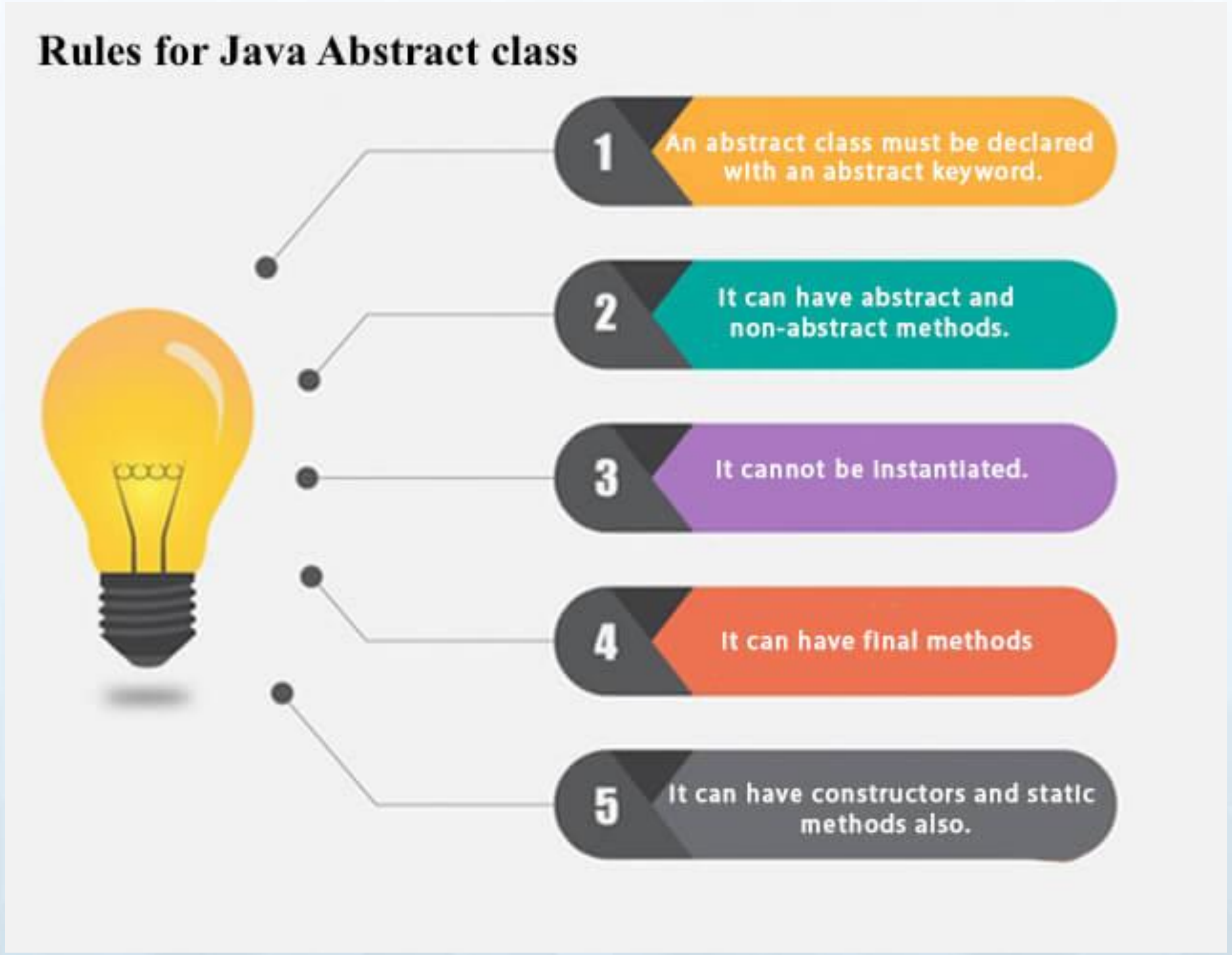
- **Abstract** keyword is used to create an abstract class in java.
- Abstract class in java can't be instantiated.
- Abstract keyword is used to create an abstract method, an abstract method doesn't have body.
- If abstract class doesn't have any method implementation, its better to use interface because java doesn't support multiple class inheritance.

Abstract Class Contd…

- The subclass of abstract class in java must implement all the abstract methods unless the subclass is also an abstract class.
- All the methods in an interface are implicitly abstract unless the interface methods are static or default.
- Java Abstract class is used to provide common method implementation to all the subclasses or to provide default implementation.

Note: Static methods and default methods in interfaces are added in Java 8.

Abstract Class Contd…



Rules for Java Abstract class

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.
4. It can have final methods
5. It can have constructors and static methods also.

Abstract class vs Interface

- **Type of methods:** Interface can have only abstract methods. Abstract class can have abstract and non-abstract methods. From Java 8, it can have default and static methods also.
- **Final Variables:** Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.
- **Type of variables:** Abstract class can have final, non-final, static and non-static variables. Interface has only static and final variables.
- **Implementation:** Abstract class can provide the implementation of interface. Interface can’t provide the implementation of abstract class.

Abstract class vs Interface

- **Inheritance vs Abstraction:** A Java interface can be implemented using keyword "implements" and abstract class can be extended using keyword "extends".
- **Multiple implementation:** An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.
- **Accessibility of Data Members:** Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.

Interface

Interface Contd…

- An interface in java is a blueprint of a class.
- The interface in Java is a mechanism to achieve abstraction.
- There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

Interface Contd…

- Declaring Interface

```
public interface Mobile{
public void calling();
public void texting();
}
```

Interface Contd…

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, in a similar way as a class can extend another class.

Interface Contd…

- An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Interface Contd…

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface can extend multiple interfaces.

Interface Contd…

```
interface  Mobile{
    static final String
internalMemorySize="8GB";
    public   void calling();
    public   void texting();
}
class HexaMobile  implements Mobile{

    public   void calling(){
        System.out.println("Calling by voice");
    }
public  void texting(){
        System.out.println("Animated texting");
    }
    public static void main(String[] args) {
        HexaMobile hexa=new HexaMobile();
        hexa.calling();
        hexa.texting();

System.out.println("Memory:"+internalMemo
rySize);
    }
}
```

OOPS – Case study

OOPS_CaseStudy

OOPS - Assignment

OOPS
ASsignment

HEXAWARE

Innovative Services

Passionate Employees

Delighted Customers

Thank you

www.hexaware.com