***OOPS ASSIGNMENT***

1. ENCAPSULATION-

* The wrapping up of data and functions into a single unit is known as Encapsulation.
* It is also known as the Information Hiding concept.
* The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.
* The purpose of Encapsulation is -
* The encapsulated code is more flexible and easy to change with new requirements.
* It allows modifying implemented code without breaking others code who already implemented it.
* It helps the data and codes be safe from external inheritance. It also helps to achieve security.
* *The main reason we don’t follow it is because it increases the length of code and shows shutdown execution*.

Eg-

Suppose we go to a bank and deposit 10,000 Rs for 1 year with a offer that means if we give them 10,000 Rs for a year, they will repay us 20,000 Rs that sounds good , but we never know what the bank have did with your money, have they made 10,000 profit with money or not, we never know it, that how encapsulation works.

Rules of Encapsulated Class -

1. Declare private instance variables..
2. Public Setter and Getter methods.

Eg-

public class MyBank Is the main() method compulsory or not?

**// The execution starts at the main method , It is compulsory for execution,.. if we want to compile , then we won’t need the main method…( only for compilation)**

{

private int money;

public void setMoney(int money)

{

this.money=money;

}

}

* MyBank class will act as an encapsulated class where the instance variable money has been declared as private and we have the corresponding getter and setter methods for the instance variable.

REAL TIME EXAMPLE OF ENCAPSULATION-

TELEVISION , if we see the backside of the TV, it is encapsulated , we can’t see the inner wiring etc. , we can’t see anything , it is encapsulated.

2)

DIFFERENCE BETWEEN ABSTRACTION AND ENCAPSULATION

|  |  |
| --- | --- |
| Abstraction | Encapsulation |
| 1) It focuses on relevant information by hiding unnecessary details. | 1)It hides data from the outside world. |
| 2) Abstraction hides complexity by giving us a more abstract picture. | 2) It hides the internal workings so that we can change it later. |
| 3) It hides the details at design level. | 3) It hides the details at implementation level. |
| 4) To use an abstract class, we have to inherit it using base class.  Eg-  abstract class sports  {  abstract void run(); //method  } | 4) eg- School bags are one of the most real examples . It can keep our books,pens,etc  And declare the variables of a class as private. |

EXAMPLES-

ABSTRACTION-

* Our car is a great example of Abstraction. We can start a car by turning the key or pressing the start button. We don’t need to know how the engine is getting started, what all components our car has. The car internal implementation and complex logic is completely hidden from the user.

ENCAPSULATION-

* A java bean class or a class in Java and declare variables as private, wrapping up of data and code acting on that data’s methods into a single unit.

eg-

class Main

{

private String Name;

private int Roll;

private int Age;

public int getAge()

{

return Age;

}

public String getName()

{

return Name;

}

public int getRoll()

{

return Roll;

}

public void setAge( int newAge)

{

Age = newAge;

}

public void setName(String newName)

{

Name = newName;

}

public void setRoll( int newRoll)

{

Roll = newRoll;

}

}

public class TestEncapsulation

{

public static void main (String[] args)

{

Main obj = new Main();

obj.setName("Harman");

obj.setAge(21);

obj.setRoll(90);

System.out.println(" name: " + obj.getName());

System.out.println(" age: " + obj.getAge());

System.out.println("roll: " + obj.getRoll());

}

}

|  |  |
| --- | --- |
| INSTANCE METHOD | STATIC METHOD |
| 1) Methods which require an object of it’s class to be created before it can be called. | 1)Static methods are the methods in java that can be called without creating an object of class. |
| 2) Not with a static keyword. | 2) Declared with a static keyword. |
| 3) It exists as multiple copies depending on the number of instances created for that class. | 3) It means which will exist as a single copy for a class. |
| 4) Invoked using object reference. | 4) Invoked using class reference.. |
| 5) Instance methods can access static variables and methods directly. | 5) Static methods can’t access instance methods and variables directly. |
| 6) eg- We call a repairman(instance) to fix our TV. He comes with tools(parameters). He comes with specific tools needed for fixing TV and he can fix other things also. | 6) eg - The same repairman(static). When we call him to specify which repairman to call(like electrician). He will come to fix it on TV only. But, he doesn’t have tools to fix other things. |

|  |  |
| --- | --- |
| STATIC VARIABLES | INSTANCE VARIABLES |
| 1) When a variable is declared as static, then a single copy of the variable is created and shared among all objects at a class level.  Static variables are essentially global variables. | 1) Instance variables are non-static variables and declared in a class outside any method, constructor or block.  It can be accessed by only creating objects. |
| 2) Can be accessed by static and non static methods both | 2) Cannot be accessed inside a static method. |
| 3) eg-  Is it compulsory to have one public class in java? // **no , it will not create error , it is not compulsory if we are creating a package , But if we want to compile and execute then keeping main method as public is good, (If we are declaring class without public , it will not create error ) ( Also, like encapsulation is that which hides variables from other classes, those variables can be accessed with encapsulated class methods only)**  public class A  {  static int age;  static String name;  static void display()  {  System.out.println(age);  System.out.println(name);  }  public static void main(String[] args)  {  age=20;  name= “Harman”;  display();  }  }  Output:  20  Harman | 3) eg-  public class A  {  public String name;  private int marks;  public A(String Name)  {  name= Name;  }  public void Marks(int Mark)  {  marks=Mark;  }  public void getMark()  {  System.out.println(name);  System.out.println(marks);  }  public static void main(String[] args)  {  A a1=new A(“Harman”);  A a2= new A(“Harry”);  A a3=new A(“Harish”);  a1.Marks(95);  a2.Marks(93);  a3.Marks(90);  a1.getMark();  a2.getMark();  a3.getMark();  }  }  Output:  Harman  95  Harry  93  Harish  90 |

4)It is a process in which a call to an overridden method is resolved at runtime rather than compile time or Whenever an object is bound with functionality at runtime , this is known as Runtime Polymorphism.

* It is achieved by method overriding. It can be achieved by static properties like a static method inside a class.
* eg-

public class Animal

{

public void run()

{

System.out.println(“Animal is running”);

}

}

class Cat extends Animal

{

public void run()

{

System.out.println(“meow”);

}

public static void main(String[] args)

{

Animal obj = new Cat();

obj.run();

}

}

Output:

meow

Eg- Dynamic Polymorphism is divided into 2 parts -

1. Method overriding - Means something existing was available now that has been overridden by something new one.
2. Dynamic Dispatch -

This is the mechanism by which a call to an overridden method is resolved at run time.We always need instance properties of a class to achieve RTP.

Eg-(METHOD OVERRIDING)

Class Animal class Dog extends Animal

{ {

void eating() void eating()

{ {

} }

void sleeping()

{

}

void walking()

{

}

}

5) METHOD SIGNATURE-

* It is a method or a list of arguments which is identified by its - unique method signature, which usually includes the method name, and the number, types and order of its parameters.
* It is the smallest type of a method.
* Eg-

public int m1(int i, float f)

m1(int,float) //Method\_signature

* Eg-

1. class Test

{

public void m1(int i)

{

}

public void m2(String s)

{

}

public static void main(String[] args)

{

Test t=new Test();

t.m1(10); //successfully execute

t.m2(“harman”); //successfully execute

t.m3(10.5); //gives an error because 2 methods are defined i.e. int and string.

If we give t.m3(10); will it throw an error? Describe error reason.. ..// **It would throw an error as there is no method with that name. There must be a matching method defined in the class.**

6)

|  |  |
| --- | --- |
| STATIC BINDING (early-binding) | DYNAMIC BINDING (late-binding) |
| 1) Events that occur at compile time by compiler are called static binding. | 1) Events that occur at run time are Dynamic Binding. |
| 2) All information needed to call a function is known at compile time. | 2) All information needed to call a function comes to know at run time. |
| 3) It is efficient. | 3) It is flexible. |
| 4) Fast execution, if there is any private, final or static method in a class, there is static binding. | 4) Slow execution. (Overriding) |
| 5) eg- Overloaded function call, overloaded operators. | 5) eg- Virtual functions used in C++, Overridden methods in Java |
| 6) eg-  class Person  {  private void eat()  {  System.out.println(“person is eating”);  }  public static void main(String[] args)  {  Person p1=new Person();  p1.eat();  }  } | 6) eg-  public class Dog *//Dog is a newClass*  {  public void print() *//method parent class* *will execute first*  {  System.out.println("print in superclass");  }  public static void main(String[] args)  {  Dog A=new Dog(); *//We create object of Dog and give reference of Dog class, Dog method will be invoked*  A.print();  Dog B= new C(); *//reference of Dog*  B.print();  }  }  class C extends Dog *//C is a subnewclass and Dog is a newclass*  {  public void print()  *//here we overridden print method in subnewclass*  {  System.out.println("print in subclass"); **//method overriding  *//****second time, child method will be invoked.*  }  }  Output-  print in superclass  print in subclass |

7) DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE

|  |  |
| --- | --- |
| ABSTRACT CLASS | INTERFACE |
| 1) Abstract keyword contain non-final variables as variables are declared as non final. | 1) Interface keyword contain final,public and static. |
| 2) Subclasses **extends** abstract class. | 2) Subclasses **implements** interfaces. |
| 3) We can extend only one abstract class, and do not support multiple inheritance. | 3) We can implement multiple interfaces. |
| 4) Static and Non static variable. | 4) Static variable. |
| 5) All new modifiers. | 5) Only public access specifiers. |
| 6) Contains constructors. | 6) Doesn’t contain constructors. |
| 7) Abstract classes should be used for objects that are closely related. | 7) Interfaces are best suited for providing common functionality to unrelated classes. |
| 8) eg-  abstract class Scooter  {  abstract void run();  }  class Activa extends Scooter  {  void run()  {  System.out.println(“ “);  }  public static void main(String[] args)  {  Scooter obj = new Activa();  obj.run();  Not running  <?php  abstract class a  {  abstract public function display1();  abstract public function display2();  }  class b extends a  {  public function display1()  {  echo " Harmanjit ";  }  public function display2()  {  echo " Singh ";  }  }  $obj = new b();  $obj->display1();  $obj->display2();  ?> | 8) eg-  interface interface\_name  interface inter  {  final int a = 5;  void get();  }  class Test implements inter  {  public void get()  {  System.out.println(“Webners”);  }  public static void main(String[] args)  {  Test t= new Test();  t.get();  System.out.println(a);  }  Write complete example  public class Ant implements Animal  {  public void eat()  {  System.out.println(“Ant eats”);  }  public void travel()  {  System.out.println(“Ant travels”);  }  public int legs()  {  return 0;  }  public static void main(String[] args)  {  Ant a = new Ant();  a.eat();  a.travel();  }  } |

8) DIFFERENCE BETWEEN PRIVATE AND INNER CLASS

|  |  |
| --- | --- |
| PRIVATE CLASS | INNER CLASS |
| 1) A private class cannot be accessed from an object outside the class and it can be accessed by itself and by default all members and variables will be private. | 1) Inner class is like a nested class(means class within a class), it can be extended . An inner class can be private, it cannot be accessed from an object outside the class, an object can also be created. |
| 2)Private classes are very important because control of scope is important | 2)Inner classes are useful because if a class is useful to only one class, it makes nested and together, it helps in packaging of the classes. |
| 3) We cannot declare a top level class as private. It can be either public or default. | 3) Java inner class implements encapsulation.Can be declared within a method of an outer class. |

* Inner classes help in better maintenance of the code.

# public class OuterClass

{

//outer class code

public class InnerClass

{

//inner class code

}

}

Eg-

class Outer

{

void outerMet()

{

final int x=50;

System.out. println(“inside outerMet”);

class Inner

{

void innerMet()

{

System.out. println(“x= “+x);

}

}

Inner z=new Inner();

z.innerMet();

}

}

class Demo

{

public static void main(String[] args)

{

Outer x = new Outer();

x.outerMet();

}

}

Private class Example- ( Java follows oops concepts. If we make any class as private, it will not be accessible from another class. So there can't be inheritance, runtime polymorphism, abstraction etc. but a nested class can be private)

eg-

class A

{

private class B

{

void msg()

{

System.out.println("hello");

}

}

public static void main(String args[])

{

A.B b=new A().new B();

b.msg();

}

}

Output:

hello

9) **COMPOSITION-**

* It is one of the fundamental concepts in OOP.
* If an object contains the other object and contained object cannot exist without the existence of that object, then it is called Composition.
* It describes a class that references one or more objects of other classes in instance variables.
* This allows us to model a **has-a** association between objects.
* It is a specialized form of aggregation. In composition, if the parent object is destroyed, then the child objects also exist.
* It is a strong aggregation called “death” relationship.

**Benefits of Composition-**

1. Reusing existing code
2. Design API.
3. We can control the visibility of other objects to classes and reuse only what we need.
4. It also allows creation of backend classes when needed.

eg- Composition example like School and teacher. We will create a class *School* that contains data members like students, section and another class *Teacher* that has a reference to refer to the list of classrooms. A teacher can have a number of classrooms in different sections.

import java.io.\*;

import java.util.\*;

class school

{

public String classroom;

public String section;

school(String classroom,String section)

{

this.classroom=classroom;

this.section=section;

}

}

class Teacher

{

private final List<school> schools;

Teacher(List<school> schools)

{

this.schools=schools;

}

public List<school> getTotalTeacherInSchools()

{

return schools;

}

}

class A

{

public static void main(String[] args)

{

school s1=new school(“Kindergarten”,”A”);

school s2=new school(“LKG”,”B”);

school s3=new school(“UKG”,”C”);

List<school> school=new ArrayList<school>();

school.add(s1);

school.add(s2);

school.add(s3);

Teacher teacher = new Teacher(school);

List<school> schools= Teacher.getTotalTeacherInSchools();

for(school sl : schools)

{

System.out. println(“Classroom : “ + sl.classroom +” and” + ”Section : “ + sl.section);

}

}

}

Output:

Classroom: Kindergarten: A

Classroom: LKG: B

Classroom: UKG: C

10)  **PHP EXAMPLES OF ENCAPSULATION-**

<?php

class A

private id;

private password;

public function updatepassword(id, password)

{

echo(“update password”.$password.”.$id);

echo “<br>”;

}

public function name($id)

{

echo “<br>”;

}

}

$obj=new A();

$obj->updatepassword(‘abc’,’xyz’);

$obj-> name(‘abcd’);

?>

1)

//

<?php

class User

{

private $id;

private $name;

private $email;

//setter methods

public function setId($id)

{

$this.->id=id;

//getter method

public function getId()

{

return $this->id;

}

//setter method

public function setName($name)

{

$this.->name=$name;

}

//getter method

public function getName()

{

return $this->name;

}

//setter method

public function setEmail($email)

{

$this->email=$email;

}

//getter method

public function getEmail()

{

return $this->email;

}

}

2)

class Person {

// private field

private int age;

// getter method

public int getAge() {

return age;

}

// setter method

public void setAge(int age) {

this.age = age;

}

}

public class Main {

public static void main(String[] args) {

// create an object of Person

Person p1 = new Person();

// change age using setter

p1.setAge(21);

// access age using getter

System.out.println("My age is " + p1.getAge());

}

}

Output-

My age is 21

**ACCESS MODIFIERS**

There are 3 modifiers in Php oop-

1)Public

2)Protected

3)Private

By default, all members are treated as public members. If we specify public in front of class name in php oop, it will give a parse error.

Eg-

<?php

class A

{

public $name=”Harman”;

private $password=’harmann’;

protected $email=’[harmanjit.singh@webners.com](mailto:harmanjit.singh@webners.com)’’

public function getPass()

{

return $this->password;

}

}

class B extends A

{

function\_construct()

{

echo $this->email;

}

2) <html>

<body>

<?php

class Fruit {

public $name;

protected $color;

private $weight;

}

$mango = new Fruit();

$mango->name = 'Mango'; // OK

$mango->color = 'Yellow'; // ERROR

$mango->weight = '300'; // ERROR

?>

</body>

</html>

**PHP INHERITANCE-**

<?php

class Vehicle

{

public $name;

public $color;

public function\_construct($name,$color)

{

$this->name=$name;

$this->color=$black;

}

public function get()

{

echo “Vehicle is {$this->name} and the color is {$this->color}”;

}

}

Class Car extends Vehicle

{

public function message()

{

echo “is the car red or black?”;

}

}

$car=new Car(“Car”,”black”);

$car->message();

$car->get();

?>

Output-

is the car red or black? Vehicle is Car and the color is black.

**PHP Interface-**

<?php

interface Animal

{

public function sound();

}

Class Dog implements Animal

{

public function sound()

{

echo “bhow”;

}

}

$animal=new Dog();

$animal->sound();

?>

Output-

bhow

**PHP ABSTRACT CLASSES-**

An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented on the code.

eg-

<?php

abstract class ParentClass

{

public $name;

abstract protected function calc($a,$b);

}

class childClass extends parentClass

{

public function calc($c,$a)

{

echo “Hi”;

}

}

$test=new childClass();

$test=calc(10,20);

?>