Overloading Overriding Runtime Type and Object Orientation

19) State the benefits of encapsulation in object oriented design and write code that implements tightly encapsulated classes and the relationships "is a" and "has a".

- The collective term for "datatype and operations bundled together, with access restrictions" is a class. The individual elements in a class are fields.
- A variable of some class type is called an instance or an object. It is same as a variabe of an ordinary type, except the valid functions that can be invoked for it are tied to it.

"is a" versus "has a"

Everything is an object. For example, buildings, furniture, animals etc. A class is made up of objects. Any object belonging to this class will share these characteristics and behaviors.
Think of an object as a unit that can be made up of other smaller units. Think of an apartment building, for example. It is made up of number of apartments. Each apartment has doors, windows, a kitchen, a bedroom, and a bathroom. Apartments are the objects in this example. The apartment building is the class. It is made up of objects, or units. These objects are not all exactly alike, but they have enough similar characteristics that they can be classed together.

To distinguish between "is a" and "has a", take the above example.
Now, how would you decide whether to use inheritance or nesting to associate the two.
Would you say "apartment has a door" or "apartment is a door"?
Your answer should be "has a", so use nesting.

Would you say "apartment is a building" or "apartment has a building"?
Your answer should be "is a", so use inheritance to extend the building class in the apartment class.

Encapsulation :

One of the important object−oriented techniques is "encapsulation". The technique of hiding the data within the class, and making it available only through the methods is known as encapsulation, because it seals the class's data safely inside the "capsule" of the class, where it can be accessed only by trusted users−−i.e., by the methods of the class.

eg,

```
class building {
private int no_of_apts=30;
protected int bedroom;
protected int bathroom;
public void num_of_doors(int door) {
int num = door;
System.out.println("Total number of doors : " + num);
}
public int setMethod() {
return no_of_apts;
}
}

class apartment extends building {
```

```
int apt_number;
int resident_name;
public void num_of_doors(int door) {
int num = door;
System.out.println("Number of doors in an apt : " + num);
}

}

public class inherit_test {
public static void main(String arg[]) {
building b = new building();
apartment a = new apartment();
System.out.println(b.no_of_apts); //compile error :
//Variable no_of_apts in class building not accessible from class
//inherit_test
int apts=b.setMethod(); //data is accessed thro' a method and not directly.
//It is encapsulated.
System.out.println("Number of Apartments : " + apts);
b.num_of_doors(15);
a.num_of_doors(2);
}
}
```

Remember :

- To identify classes look for nouns and to identify methods look for verbs in the problem analysis. eg, class − apartment, building; method − counting number of doors
- Inheritance ("is a") is for specialization of a type, and container ("has a") classes are for code re−use.

---

20) Write code to invoke overridden or overloaded methods and parental or overloaded constructors; and describe the effect of invoking these methods.

A method is a name for an action. You refer to all objects and methods by using names. Well−chosen names make it easier for you and others to understand your code.

Overloading :

Say, there is a class called Drink and a subclass called hotDrink. For our example we will take coffee and tea. Both has a preparation method, but the ingredients and outcome are different.

Overloading is almost a must for constructors. Say, you want to create an object in more than one way, ie, you need two constructors, one that takes no argument, and one that takes an int argument, and there can be only one constructor name as constructor's name is predetermined by the name of the class, you need method overloading.

eg,

```
class A {
int a;
A() {
```

```
a=0;
}
A(int i) {
a=i;
System.out.println(a);
}
public void method() {
}//no parameters, return type is void

public void method(String[] c) {
}//one parameter, return type is void

public int method(int a, int b) {
}//two parameters, return type is int
}
public class overload {
public static void main(String[] arg) {
A aClass = new A();
A bClass = new A(5);
aClass.method();
bClass.method("hi");
aClass.method(5,6);
}
}
```

eg using this keyword,

```
class A {
public A(String a) {
this(int i, int j);
}
public A( int m, int n) {
int k=m;
int l=n;
}
.....
}
```

when you call the constructor new A("hi"), then the A(String) constructor calls the A(int, int) constructor. This will be useful when there are common code between constructors.

As overloaded methods take a unique list of argument types, Java can easily find which method to call.

Overriding :

- When a class defines a method using the same name, return type, and arguments as a method in its superclass, the method in the class overrides the method in the superclass.
- When the method is invoked for an object of the class, it is the new definition of the method that is called, and not the method definition from superclass.
- Methods may be overridden to be more public, not more private.

Say, there is a class called hotDrink and a subclass called tea. tea can be cold or hot. Though you can use the same ingrediants the preparation differs. So you have to implement new versions of these functions.

eg see also the use of the keyword super,

```
class overridden {
int i = 5;
public void method() {
System.out.println(i);
}
}

class override extends overridden {
int i=10;
public void method() {
System.out.println("i in override : " + i);
i = super.i;
System.out.println("i in overridden : " + i);
}
public static void main(String s[]) {
override o = new override();
o.method();
}
}
```

output :

i in override : 10
i in overridden : 5

Overloaded	*Overridden*
supplements each other	(largely) replaces the method it overrides
can exist in any number within same class	Each method in a parent class can be overridden atmost once in any one class
must have different arguments	must have argument list of identical type and order
return type may be freely chosen	return type must be identical
determined at compile time	determined at runtime

Remember :

- Any type may be returned, only the number, order and types of parameters are considered and should be different for overloading. Ofcourse, the name of the method should be same
- Argument list's type, order and return type, should be identical for overriding.

21) Write code to construct instances of any concrete class including normal top level classes inner classes static inner classes and anonymous inner classes.

You can create four different types of inner classes, based upon the how and the where of creation.

- Nested top−level classes
  - ♦ If you declare a class within a class and specify the static modifier, the compiler treats the class just like any other top−level class.
  - ♦ Any class outside the declaring class accesses the nested class with the declaring class name acting similarly to a package. eg, outer.inner
  - ♦ Top−level inner classes implicitly have access only to static variables.
  - ♦ There can also be inner interfaces. All of these are of the nested top−level variety.
- Member classes
  - ♦ Member inner classes are just like other member methods and member variables and access to the member class is restricted, just like methods and variables. This means a public member class acts similarly to a nested top−level class.
  - ♦ The primary difference between member classes and nested top−level classes is that member classes have access to the specific instance of the enclosing class.
- Local classes
  - ♦ Local classes are like local variables, specific to a block of code.
  - ♦ Their visibility is only within the block of their declaration.
  - ♦ In order for the class to be useful beyond the declaration block, it would need to implement a more publicly available interface.
  - ♦ Because local classes are not members, the modifiers public, protected, private, and static are not usable.
- Anonymous classes
  - ♦ Anonymous inner classes extend local inner classes one level further.
  - ♦ As anonymous classes have no name, you cannot provide a constructor.

eg,

```
public class MyPublicClass {
        public MyPublicClass() { ... }

        class MyHelperClass {
        MyHelperClass() { ... }
        int someHelperMethod(int z, int q)
        { ... }
        }

        void method1() { ... }
        int method2() { ... }
        public method3(int x, int y) { ... }
        }
```

here, there are 2 classes. One is called MyPublicClass and the other is mapped to the name MyPublicClass$MyHelperClass.

another eg,

```
public class OuterClass {
```

```
int outerx = 10;
class InnerClass {
public InnerClass() {
System.out.println("outer = " + outerx);
}
}

InnerClass inner;
public void createInner() {
inner=new InnerClass();
}
public static void main(String arg[]) {
OuterClass outer = new OuterClass();
outer.createInner();
}
}
```

to reference the inner class from another class

```
class outer {
public class inner {
//methods variables
}
//methods variables
}

class other {
public static void main(String arg[]) {
outer.inner myInn = new outer().new inner();
}
}
```

Accessibility :

```
public class outer {
public int a=1;
static int b=5;
final int c=10;
private int g=25;
public static void main(String s[]) {
final int k=10;
// int l=25;
class Inner {
int d=3;
final int e=15;
private int h=25;
// static int f=20;//can't be static in local class
public void print() {
outer o=new outer();
System.out.println("a " + o.a);
System.out.println("b " + b);
System.out.println("c " + o.c);
```

```
System.out.println("g " + o.g);
System.out.println("k " + k);
// System.out.println("l " + l);//attempt to use a non−final
variable
}
}
Inner in=new Inner();
in.print();
System.out.println("d " + in.d);
System.out.println("e " + in.e);
System.out.println("h " + in.h);
System.out.println("i " + in.i);
}
}
}
```

- Since Inner is not static, it has reference to an enclosing object and all variables of that object are accessible.
- varaibles in enclosing method are accessible only if marked "final"

Java Term	***Description***	***Example***
Nested Top−Level class	♦ Nested Top−Level class is a nested class that is declared "static" ♦ Nested top−level classes are used as a convenient way to group related classes. ♦ This is not an inner class, but a new kind of top−level class ♦ Since static doesn't have a "this" pointer to an instance of the enclosing class, a nested class has no access to the instance data of objects for its enclosing class. ♦ Any class outside the declaring class accesses the nested class with the declaring class name acting similarly to a package.(eg, If class "top" has nested top−level class called "myNested", this would be referred to as top.myNested) ♦ Top−level inner classes implicitly have access only to static variables.	*class outer {* *int a, b;* *static class myInne* *int c, d;* *void innerMethod(* *}* *void outerMethod(* *myInner mi = new* *}* *}* *class other {* *outer.myInner outI* *outer.myInner();* *}*
Member class	♦ A nested class cannot have any "static" varaibles inside. ♦ This is an inner class ♦ Scope of the inner class is the entire parent in which it is directly nested. ie, the inner class can reference any members in its parent. ♦ The parent must declare an instance of an inner class before it can invoke the inner class methods, assign to data fields and so on (including private ones) ♦ Unlike nested top−level classes, inner classes are not directly part of a package and are not visible outside the class in which they are nested.	*class outer {* *int a=5;* *class myInner {* *int c=a;* *void innerMethod(* *}* *void outerMethod(* *myInner mi1 = new* *myInner mi1 = new* *mi1.c = mi2.c + 30* *}*

		}
Local class	♦ This is an inner class declared within a block, typically within a method ♦ It is not a member of the enclosing class. ♦ Their visibility is only within the block of their declaration. ♦ In order for the class to be useful beyond the declaration block, it would need to implement a more publicly available interface. ♦ Because local classes are not members, the modifiers public, protected, private, and static are not usable. ♦ Local classes can access only final varaibles or parameters.	*public class myApp* *JApplet {* *JButton b = new* *JButton("click");* *public void init() {* *class myInnerBH i* *ActionListener {* *public void* *actionPerformed(A* *e) {* *System.out.println(* *pressed");* *}* *}* *add(b);* *b.addActionListene* *myInnerBH());* *}* *}*
Anonymous class	♦ A variation on a local class. The class is declared and instantiated within a single expression ♦ These classes are simply inner classes that are not given a specific name. ♦ When you don't even need a name because you really just want to pass a method that does something, then you can get away with creating an anonymous inner class. ♦ Typically, a class is not named when it is used only once. ♦ We are declaring an object of an anonymous class type that either implements the interface or extends the class. If it extends the class, then any methods we define may override the corresponding methods of the base class.	*public class A exte* *{* *JButton b = new* *JButton("click");* *public void init() {* *b.addActionListene* *ActionListener() {* *public void* *actionPerformed(A* *e) {* *System.out.println(* *performed");* *}* *});* *}*

Remember :

- An inner class cannot have the same name as its enclosing class.
- An inner class can extend any class or interface.