CHP-23
Java Abstraction

Abstraction refers to the ability to make a class abstract in OOP. An abstract class is one that cannot be instantiated. All other functionality of the class still exists, and its fields, methods, and constructors are all accessed in the same manner. You just cannot create an instance of the abstract class.
If a class is abstract and cannot be instantiated, the class does not have much use unless it is subclass. This is typically how abstract classes come about during the design phase. A parent class contains the common functionality of a collection of child classes, but the parent class itself is too abstract to be used on its own.

Abstract Class:

Use the **abstract** keyword to declare a class abstract. The keyword appears in the class declaration somewhere before the class keyword.

```
/* File name : Employee.java */
public abstract classEmployee
{
private String name;
private String address;
private int number;
public Employee(String name,String address,int number)
{
System.out.println("Constructing an Employee");
this.name = name;
this.address = address;
this.number = number;
}
public double computePay()
{
System.out.println("Inside Employee computePay");
return0.0;
}
public void mailCheck()
{
System.out.println("Mailing a check to "+this.name
+" "+this.address);
}
public String toString()
{
return name +" "+ address +" "+ number;
}
```

CHAPTER

```
public String getName()
```

```
{
return name;
}
public String getAddress()
{
return address;
}
public void setAddress(String newAddress)
{
address = newAddress;
}
public int getNumber()
{
return number;
}
}
```

Notice that nothing is different in this Employee class. The class is now abstract, but it still has three fields, seven methods, and one constructor.
Now if you would try as follows:

```
/* File name : AbstractDemo.java */
public class AbstractDemo
{
public static void main(String[] args)
{
/* Following is not allowed and would raise error */
Employee e =new Employee("George W.","Houston, TX",43);
System.out.println("\n Call mailCheck usingEmployee reference--");
e.mailCheck();
}
}
```

When you would compile above class, then you would get the following error:

```
Employee.java:46:Employee is abstract; cannot be instantiated
Employee e =new Employee("George W.","Houston, TX",43);
^
1 error
```

Extending Abstract Class:

We can extend Employee class in normal way as follows:

```
/* File name : Salary.java */
public class Salary extends Employee
{
private double salary;//Annual salary
public Salary(String name,String address,int number,double salary)
{
super(name, address, number);
setSalary(salary);
}
public void mailCheck()
{
System.out.println("Within mailCheck of Salary class ");
System.out.println("Mailing check to "+ getName()
```

```
+" with salary "+ salary);
}
public double getSalary()
{
return salary;
}
public void setSalary(double newSalary)
{
if(newSalary >=0.0)
{
```

```
salary = newSalary;
}
}
public double computePay()
{
System.out.println("Computing salary pay for "+ getName());
return salary/52;
}
}
```
Here, we cannot instantiate a new Employee, but if we instantiate a new Salary object, the Salary object will inherit the three fields and seven methods from Employee.
```
/* File name : AbstractDemo.java */
public class AbstractDemo
{
public static void main(String[] args)
{
Salary s =new Salary("Mohd Mohtashim","Ambehta, UP",
3,3600.00);
Employee e =new Salary("John Adams","Boston, MA",
2,2400.00);
System.out.println("Call mailCheck using Salary reference --");
s.mailCheck();
System.out.println("\n Call mailCheck usingEmployee reference--");
e.mailCheck();
}
}
```
This would produce the following result:
```
Constructing an Employee
Constructing an Employee
Call mailCheck using Salary reference --
Within mailCheck of Salary class
Mailing check to MohdMohtashim with salary 3600.0
Call mailCheck using Employee reference--
Within mailCheck of Salary class
Mailing check to JohnAdams with salary 2400.
```
Abstract Methods:
If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as abstract.
The abstract keyword is also used to declare a method as abstract. An abstract method consists of a method signature, but no method body.

Abstract method would have no definition, and its signature is followed by a semicolon, not curly braces as follows:
```
public abstract class Employee
{
private String name;
private String address;
private int number;
public abstract tdouble computePay();
//Remainder of class definition
}
```
Declaring a method as abstract has two results:
 The class must also be declared abstract. If a class contains an abstract method, the class must be abstract as well.
 Any child class must either override the abstract method or declare itself abstract.
A child class that inherits an abstract method must override it. If they do not, they must be abstractand any of their children must override it.
Eventually, a descendant class has to implement the abstract method; otherwise, you would have a hierarchy of abstract classes that cannot be instantiated.
If Salary is extending Employee class, then it is required to implement computePay() method as follows:
```
/* File name : Salary.java */
public class Salary extends Employee
```

```
{
privatedouble salary;// Annual salary
public double computePay()
{
System.out.println("Computing salary pay for "+ getName());
return salary/52;
}
//Remainder of class definition
}
```

CHP-24
Java Encapsulation

Encapsulation is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.
Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding.
Encapsulation can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class. Access to the data and code is tightly controlled by an interface.
The main benefit of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code. With this feature Encapsulation gives maintainability, flexibility and extensibility to our code.

Example:

Let us look at an example that depicts encapsulation:

```
/* File name : EncapTest.java */
public class EncapTest{
private String name;
private String idNum;
private int age;
public int getAge(){
return age;
}
publicString getName(){
return name;
}
publicString getIdNum(){
return idNum;
}
publicvoid setAge(int newAge){
age = newAge;
}
publicvoid setName(String newName){
```

CHAPTER

```
name = newName;
}
public void setIdNum(String newId){
idNum = newId;
}
```

```
}
```
The public methods are the access points to this class' fields from the outside java world. Normally, these methods are referred as getters and setters. Therefore any class that wants to access the variables should access them through these getters and setters.
The variables of the EncapTest class can be access as below:
```
/* File name : RunEncap.java */
public class RunEncap{
public static void main(String args[]){
EncapTest encap =new EncapTest();
encap.setName("James");
encap.setAge(20);
encap.setIdNum("12343ms");
System.out.print("Name : "+ encap.getName()+" Age : "+ encap.getAge());
}
}
```
This would produce the following result:
```
Name:JamesAge:20
```
Benefits of Encapsulation:
The fields of a class can be made read-only or write-only.
A class can have total control over what is stored in its fields.
The users of a class do not know how the class stores its data. A class can change the data type of a field and users of the class do not need to change any of their code.