



National University of Computer & Emerging Sciences,
Karachi



Computer Science Department
Spring 2022, Lab Manual – 08

Course CL-217	Code:	Course : Object Oriented Programming Lab
Instructor(s) :		Abeer Gauher, Hajra Ahmed, Shafique Rehman

LAB - 8

Abstract Classes & Interface

Abstract Class:

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. It cannot be instantiated.

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Syntax

```
abstract class A{
```

Abstract Method:

A method which is declared as abstract and does not have implementation is known as an **abstract method**.

Syntax

```
abstract void myFunction( ); //no method body and abstract
```

Example

```
abstract public class Vehicle {  
    abstract public void run(); // abstract method  
}
```

It is the responsibility of child class(es) to override the abstract function and “complete” the parent class.

Example

```
class Car extends Vehicle{  
    public void run(){  
        System.out.println("Running Fast");  
    }  
}
```

Example: (Using abstract class from main)

```
public class Main
{
    public static void main(String[] args) {
        Vehicle obj = new Car();
        obj.run();
    }
}
```

Output:

```
Running Fast
...Program finished with exit code 0
Press ENTER to exit console.
```

Point to Remember: We cannot create an instance of the abstract class.

```
abstract class Animal {
    abstract public void eat(); // abstract method
}

public class Main
{
    public static void main(String[] args) {
        Animal obj = new Animal(); // trying to creating the insance
                                   //of abstract class
        obj.eat();
    }
}
```

Compilation failed due to following error(s).

```
Main.java:10: error: Animal is abstract; cannot be instantiated
    Animal obj = new Animal();
                  ^
```

Point to Remember: An abstract class can contain concrete functions.

Example:

```
abstract class Animal {
    abstract public void eat(); // abstract method
    public void mov(){
        System.out.print("Animal is moving"); // concrtese method
    }
}
```

```
public class Main
{
    public static void main(String[] args) {
        Animal obj = new Dog();
        obj.mov();
    }
}
```

Animal is moving

Note: If there is an abstract method in a class, that class must be abstract.

Example:

```
class Animal {
    abstract public void eat(); // abstract method
}
class Dog extends Animal{
    public void eat(){
        System.out.println("Animal is Eating ");
    }
}
public class Main
{
    public static void main(String[] args) {
        Animal obj = new Dog();
        obj.eat();
    }
}
```

Main.java:1: error: Animal is not abstract and does not override abstract method eat() in Animal
class Animal {
^
1 error

Anonymous Class:

If we do not have a child class, we can still override the abstract function by creating an **Anonymous Class**.

- A new class is defined (without a name, so called anonymous class)
- This new class extends abstract base class
- Abstract methods are overridden in this new class
- New instance of this new class is created and assigned to the parent variable

Example

```
abstract class Animal {
    abstract public void eat(); // abstract method
}
public class Main
{
    public static void main(String[] args) {
        Animal obj = new Animal() {
            public void eat(){
                System.out.println("Animal is eating...");
            }
        }; // end of Anonymous class
        obj.eat();
    }
}
```

Animal is eating...

Interfaces:

An **interface** in Java is a blueprint of a class. It has static constants and abstract methods. A difference between abstract classes and interface is that there can be concrete methods in abstract classes whereas interface cannot contain any.

How to create an interface:

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
interface <interface_name>{  
  
    // declare constant fields  
    // declare methods that are abstract  
    // by default.  
}
```

Example: (Declaring an interface)

```
interface printable{  
    public void print();  
}
```

Example: (Implementing interface in class)

```
class Test implements printable{  
    public void print(){  
        System.out.println(" I was declared in interface and I am implemented in class");  
    }  
}
```

Points to Remember: We can create static methods in interfaces. But we must define their implementation inside the interface.

Example:

```
interface printable{  
    public static void print(){  
        System.out.println("I am static method of Interface");  
    }  
}
```

Points to Remember: When a class implements an interface, we can save the object of that class in the interface variable.

Example:

```
public class Main
{
    public static void main(String[] args) {
        printable obj = new Test();
        obj.print();
    }
}
```

I was declared in interface and
I am implemented in class

Lab Exercise

1. Create an abstract class 'Bank' with an abstract method 'getBalance'. \$100, \$150 and \$200 are deposited in banks A, B and C respectively. 'BankA', 'BankB' and 'BankC' are subclasses of class 'Bank', each having a method named 'getBalance'. Call this method by creating an object of each of the three classes.
2. We have to calculate the percentage of marks obtained in three subjects (each out of 100) by student A and in four subjects (each out of 100) by student B. Create an abstract class 'Marks' with an abstract method 'getPercentage'. It is inherited by two other classes 'A' and 'B' each having a method with the same name which returns the percentage of the students. The constructor of student A takes the marks in three subjects as its parameters and the marks in four subjects as its parameters for student B. Create an object for each of the two classes and print the percentage of marks for both the students.
3. Define an abstract class **Character** that contains:
 - an integer health variable that represents the character's health,
 - an integer attackPower variable that represents the character's attack power
 - string name variable that represents the character's name.
 - specialAbility() (abstract method): An abstract method that must be implemented by the subclass for each character type. This method defines the unique special ability of each character type.
 - attack() (non-abstract method): A method that takes another Character object as a parameter and simulates an attack from the current character to the target character. It subtracts the attack power of the current character from the target character's health.

- defend() (non-abstract method): A method that simulates the current character defending itself.

Define **Warrior** class that contains:

- specialAbility() (method implementation): Overrides the abstract specialAbility() method in the Character class to define the unique special ability for a Warrior character. In this implementation, the warrior's attack power is doubled.

Define **Mage** class that contains:

- specialAbility() (method implementation): Overrides the abstract specialAbility() method in the Character class to define the unique special ability for a Mage character. In this implementation, the mage's attack power is increased by 20.

Define class **Archer** that contains:

- specialAbility() (method implementation): Overrides the abstract specialAbility() method in the Character class to define the unique special ability for an Archer character. In this implementation, the archer's attack power is increased by 15.

Finally, create an instance of each subclass in Game(Main Class) class and set their attributes such as name, health, and attackPower. Then call the attack() and defend() methods to simulate a battle between the characters. Also call the specialAbility() method to use the unique special ability of each character.

4. Create an abstract class 'Animals' with two abstract methods 'cats' and 'dogs'. Now create a class 'Cats' with a method 'cats' which prints "Cats meow" and a class 'Dogs' with a method 'dogs' which prints "Dogs bark", both inheriting the class 'Animals'. Now create an object for each of the subclasses and call their respective methods.
5. Define an interface called MessageService. The interface includes two methods:
 - o send(String message, String recipient): This method takes two string values as parameters, representing the message content and the recipient's username. It sends the message to the recipient.
 - o receive(): This method receives a message from the messaging service and returns it as a string value.

Create a WhatsApp class that implements interface and provides concrete implementations for methods in interface. In the Main class, create an instance of WhatsApp, and ask the user to enter the message he wants to send also enter the name to whom he wants to send a message and print it.

6. Define a interface called BankAccount. The interface includes three methods:

- `deposit(double amount)`: This method takes a double value as a parameter and deposits amount in user account.
- `withdraw(double amount)`: This method takes a double value as a parameter and withdraws the amount from user accounts.
- `getBalance()`: This method returns the current balance of the user's account as a double value.

Create the `SavingsAccount` class implements interface and which have double private instance variable named `balance` and provides concrete implementations for methods of interface.

In the `Main` class, create an instance of `SavingsAccount`. Your program displays the menu to user. 1 for deposit, 2 for withdraw and 3 to get total balance. And 4 for exit the application.