Java - Introduction to Programming
Lecture 12

Abstraction

Abstraction is a fundamental concept in object-oriented programming (OOP) that hides unnecessary details and exposes only the essential features to the user.

Abstraction is achieved in 2 ways:

- Abstract class
- Interfaces (Pure Abstraction)

1. Abstract Class

Abstract class is a class that cannot be instantiated by itself; it needs to be subclassed by another class to use its properties. An abstract class is declared using the “abstract” keyword in its class definition**.**

- 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.

*Note: An abstract is a Java modifier applicable for **classes** and **methods** in Java but not for **Variables**.

```
package myPackage2;

abstract class Animal {
    abstract void walk();
}

class Horse extends Animal{
    public void walk(){
        System.out.println("Walks on 4 legs");
    }
}

class Duck extends Animal{
    public void walk(){
        System.out.println("walks on 2 legs");
    }
}

public class Abstraction {
```

```
    public static void main(String[] args) {
        Horse horse = new Horse();
        horse.walk();

    }
}
```

- It cannot be instantiated.

```
package myPackage2;

abstract class Animal {
    abstract void walk();
}

class Horse extends Animal{
    public void walk(){
        System.out.println("Walks on 4 legs");
    }
}

class Duck extends Animal{
    public void walk(){
        System.out.println("walks on 2 legs");
    }
}

public class Abstraction {
    public static void main(String[] args) {
        Horse horse = new Horse();
        horse.walk();
        Animal animal = new Animal();

    }
}
```

- It can have abstract and non-abstract methods.

```
package myPackage2;

abstract class Animal {
    abstract void walk();

    public void eat(){
        System.out.println("Animal eats");
    }
```

```
}

class Horse extends Animal{
    public void walk(){
        System.out.println("Walks on 4 legs");
    }
}

class Duck extends Animal{
    public void walk(){
        System.out.println("walks on 2 legs");
    }
}

public class Abstraction {
    public static void main(String[] args) {
        Horse horse = new Horse();
        horse.walk();
        horse.eat();

    }
}
```

- It can have constructors and static methods also.

Constructor Chaining

```
package myPackage2;

abstract class Animal {
    abstract void walk();
    //constructor
    Animal(){
        System.out.println("you are creating a new animal");
    }
    public void eat(){
        System.out.println("Animal eats");
    }
}

class Horse extends Animal{
    //constructor
    Horse(){
        System.out.println("Created a horse");
    }
    public void walk(){
        System.out.println("Walks on 4 legs");
    }
```

```
}

class Duck extends Animal{
    public void walk(){
        System.out.println("walks on 2 legs");
    }
}

public class Abstraction {
    public static void main(String[] args) {
        Horse horse = new Horse();
    }
}
```

- It can have final methods, which will force the subclass not to change the body of the method.

```
package myPackage2;

abstract class Animal {
    abstract void walk();
    //constructor
    Animal(){
        System.out.println("you are creating a new animal");
    }
    final void herbyborus(){
        System.out.println("eat leaves");
    }
    public void eat(){
        System.out.println("Animal eats");
    }
}

class Horse extends Animal{
    //constructor
    Horse(){
        System.out.println("Created a horse");
    }
    public void walk(){
        System.out.println("Walks on 4 legs");
    }
}

class Duck extends Animal{
    public void walk(){
        System.out.println("walks on 2 legs");
    }
```

```
}

public class Abstraction {
    public static void main(String[] args) {
        Horse horse = new Horse();
        horse.herbyborus();
    }
}
```

2. **Interfaces**

- All the fields in interfaces are public, static and final by default.
- All methods are public & abstract by default.
- A class that implements an interface must implement all the methods declared in the interface.
- Interfaces support the functionality of multiple inheritance.
- You cannot create constructor inside interface
- All methods are abstract only

```
package myPackage2;

interface Animal {
    void walk(); //abstrct method
    void eat(){
        //non abstrct method
    }


}

public class Interface {
    public static void main(String[] args) {

    }
}
```

- You cannot create constructor inside interface

```
package myPackage2;

interface Animal {
    Animal(){

    }
```

```
}

public class Interface {
    public static void main(String[] args) {

    }
}
```

- A class that implements an interface must implement all the methods declared in the interface.

```
package myPackage;

interface Animal {
     void walk(); //abstrct method
}

class Horse implements Animal{
    public void walk(){
            System.out.println("walks on 4 legs");
        }
}

public class Interface {
    public static void main(String[] args) {
        Horse horse = new Horse();
        horse.walk();
    }
}
```

- Interfaces support the functionality of multiple inheritance.

```
package myPackage;

interface Animal {
     void walk(); //abstrct method
}

interface Herbybour {
        void eat();
}

class Horse implements Animal,Herbybour{
    public void walk(){
            System.out.println("walks on 4 legs");
```

```
        }
    public void eat(){
        System.out.println("eat leaves");
    }
}

public class Interface {
    public static void main(String[] args) {
        Horse horse = new Horse();
        horse.walk();
    }
}
```

Static Keyword

Common for all objects.

Static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)

```
package myPackage;

class Student {
    String name;
    static String schoolName;

}
public class StaticClass {
    public static void main(String[] args) {
        Student.schoolName = "DPS";

        Student student = new Student();
        student.name = "piyush";
        System.out.println(Student.schoolName);
    }
}
```

```
package myPackage;

class Student {
    String name;
    static String schoolName;

    static void pramote(){
```

```
        System.out.println("you are pramoted");
    }

}
public class StaticClass {
    public static void main(String[] args) {
        Student.schoolName = "DPS";

        Student student = new Student();
        student.name = "piyush";
        System.out.println(Student.schoolName);
        Student.pramote();
    }
}
```