



Implementing Abstraction

©2015 Manipal Global Education



INTRODUCTION

Implementing Abstraction



LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Understand Abstraction
- Create abstract classes and methods
- Define and Implement Interfaces
- Differentiate between Abstract classes and Interfaces
- Understand use of Interfaces





CONCEPT

Abstract classes



ABSTRACTION

"An Abstraction denotes the essential characteristics of an object that distinguishes it from all other kinds of objects and thus provides crisply defined conceptual boundaries, relative to the perspective of the viewer" – Grady Booch

- Concept of abstraction is implemented in java by creating abstract classes and interfaces
- Abstract classes contain the essential attributes and functionality definition's for classes of similar type
- Interfaces contain the essential functionality definition's for classes of various types sharing similar functionalities



ABSTRACT METHODS

- A class can define a method without the method implementation, such methods have to be marked as abstract
- Abstract methods are defined as shown below

```
public abstract double getArea();  
public abstract double getPerimeter();
```

- Why
 - When a class defines abstract methods, It becomes mandatory for its concrete subclasses to implement the functionality
 - Concrete subclass won't get compiled, if all the inherited abstract methods are not implemented
 - Enables a superclass to provide a service definition without specific implementation
 - Enables the methods to be accessed polymorphically



ABSTRACT CLASS

- Class having even a single abstract method has to be marked abstract as shown below

```
public abstract class Shape {  
    public abstract double getArea();  
    public abstract double getPerimeter();  
}
```

- Abstract class
 - Cannot be instantiated (object of abstract class cannot be created)
 - Can have instance and static variables, constructors, non-abstract methods (concrete methods) like any other class
 - Abstract class constructor executes through constructor chaining, when a subclass object is created



ABSTRACT CLASS AND METHOD RULES

Package :
com.manipal.abstraction
Class : Shape, Circle, Rectangle,

- It is Mandatory for the first concrete subclass to override all unimplemented abstract methods of its abstract Parent Classes

```
abstract class A {  
    abstract void m1();  
    void m2(){};  
}  
  
abstract class B extends A {  
    abstract void m3();  
}  
  
class C extends B {  
    void m1(){ //implementation }  
    void m3(){ //implementation }  
}
```

```
A obj1 = new C();  
obj1.m1();  
obj1.m2();
```

```
abstract class A {  
    abstract void m1();  
    void m2();  
}  
  
abstract class B extends A {  
    abstract void m3();  
    void m1(){ //implementation }  
}  
  
class C extends B {  
    void m3(){ //implementation };  
}
```

```
A obj1 = new C();  
obj1.m1();  
obj1.m2();
```




ABSTRACT CLASS AND METHOD RULES

- Abstract class cannot be marked final
 - Why
 - Reason for creating abstract class is that, it should be inherited
- Abstract methods cannot be made private or final
 - Why
 - Private methods are not inherited and final methods cannot be overridden
 - Violates the reason for existence of abstract methods
- It is legally allowed for an abstract class not to have any abstract method



What is the error in the following class definitions?

```
abstract class xy
{
    abstract int sum (int x, int y) {
    }
}
```

- ☐ Class header is not defined properly.
- ☐ Constructor is not defined.
- ☐ Method is not defined properly
- ☐ Method is defined properly
- ☐ No error.





1. Which of the following is FALSE about abstract classes in Java

- ☐ If we derive an abstract class and do not implement all the abstract methods, then the derived class should also be marked as abstract using 'abstract' keyword
- ☐ Abstract classes can have constructors
- ☐ A class can be made abstract without any abstract method
- ☐ A class can inherit from multiple abstract classes.





Which of the following modifier cannot be used with Abstract methods

- ☐ public
- ☐ default
- ☐ final
- ☐ protected





CONCEPT *Interfaces*



INTERFACES OVERVIEW AND ADVANTAGES

- Interface defines similarities that classes of various types share, but do not necessarily constitute a class relationship
- Interface is a contract for what a class can do, without providing the specifics of implementation
- Interface are just like classes but contain only abstract methods and constants
- Advantages
 - Provides polymorphic benefits of multiple inheritance
 - Can be implemented by any class, from any inheritance tree which share common functionality
 - Used to expose services to external application without providing specifics of implementation
 - Helps in implementing loose coupling ¹⁴



DEFINING INTERFACES

```
public interface Rewardable{  
    int calculateRewardPoints(double amount);  
}
```

- Interface can contain
 - Method prototypes
 - **only method definition** and not implementation
 - methods are implicitly **public and abstract**
 - methods must not be static/final
 - Variables
 - Are implicitly **public, static, and final**
 - Must be initialized

- Interface cannot be **instantiated**

```
Rewardable r1;           //Can be created  
r1 = new Rewardable();   // Compile Error
```



IMPLEMENTING INTERFACES

- A class can implement an interface using the implements keyword
- It should be used only after the extends keyword (if there is one)
- A class can implement more than one interface

```
public interface Rewardable{  
    int calculateRewardPoints(double amount);  
}  
  
public class SBAccount implements Rewardable{  
    public int calculateRewardPoints(double amount){  
        //implementation  
    }  
}
```

```
Rewardable r1 = new SBAccount();  
r1.calculateRewardPoints(500);
```




IMPLEMENTING INTERFACES

- A class must implement all the methods declared in the interface
- The implemented method can be marked as final

```
interface I1{  
    void m1();  
    void m2();  
}  
  
interface I2{  
    void m3();  
}  
  
class C1 implements I1,I2{  
    public void m1(){ //implementation}  
    public void m2(){ //implementation}  
    public final void m3(){ //implementation}  
}
```

```
I1 obj1 = new C1();  
obj1.m1();  
obj1.m2();
```

Using a reference of interface
only methods defined in the
interface can be accessed

```
C1 obj2 = new C1();  
obj2.m1();  
obj2.m2();  
obj2.m3();
```



ABSTRACT CLASS IMPLEMENTING INTERFACES

- An abstract class implementing an interface may choose not to provide the implementation of interface methods

```
interface I1{
    void m1();
    void m2();
}

abstract class C1 implements I1{
    abstract void m3();
    public void m2(){ //implementation}
}

class C2 extends C1{
    public void m1(){ //implementation}
    public void m3(){ //implementation}
}
```

```
I1 obj1 = new C2();
obj1.m1();
obj1.m2();

C1 obj2 = new C2();
obj2.m1();
obj2.m2();
obj2.m3();
```



INSTANCEOF

- An object of a class that implements an interface is also considered as an object of that interface

```
public interface Rewardable{...}  
  
public interface Taxable{...}  
  
public class SBAccount implements Rewardable, Taxable{...}
```

```
SBAccount sb1 = new SBAccount();  
  
System.out.println(sb1 instanceof SBAccount);    //TRUE  
System.out.println(sb1 instanceof Rewardable);    //TRUE  
System.out.println(sb1 instanceof Taxable);       //TRUE
```



EXTENDING INTERFACES

- An interface can extend one or more interfaces
- Similar to inheritance in classes
- An interface cannot implement another interface

```
interface I1{
    void m1();
}

interface I2 extends I1{
    void m2();
}

class C1 implements I2{
    public void m1(){ //implementation}
    public void m2(){ //implementation}
}
```



INTERFACE VS ABSTRACT CLASS

Interface

- A class can implement multiple interfaces
- Methods must be public and abstract
- Variables
 - must be public, static and final
 - Must be initialized
- Constructors
 - Cannot have Constructors

Abstract Class

- A class can extend only one abstract Class
- Methods have no Restrictions
- Variables
 - No Restrictions
- Constructors
 - Constructors are invoked by subclasses through constructor chaining



1. The concept of multiple inheritance is implemented in Java by

- A. Extending two or more classes.
- B. Extending one class and implementing one or more interfaces.
- C. Implementing two or more interfaces.

- ☐ B
- ☐ A and B
- ☐ B and C
- ☐ A
- ☐ C
- ☐ A, B, C





Which of these field declarations are legal within the body of an interface?

- ☐ private final static int answer = 42
- ☐ public static int answer=42
- ☐ final static answer =42
- ☐ int answer
- ☐ No error





References

- Refer following demo videos on EduNxt
 - M4I8I4 Abstract Methods And Classes – Demo
 - M4I10I2 Interfaces In Java – Demo
 - M4I10I5 Using Interfaces To Achieve Loose Coupling – Demo





SUMMARY

Implementing Abstraction



SUMMARY



In this lesson, you've learned to:

- Create Abstract classes
- Create Interfaces
- Use static polymorphism
- Use Dynamic polymorphism
- Use explicit down casting of objects in a class