OBJECT ORIENTED ANALYSIS & DESIGN
DATA STRUCTURES & ALGORITHMS

Object Oriented Programming Structure (cont.)

Constructors

Constructors

- When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

- Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Constructors (Contd.)

- Constructor is a class member function with same name as the class.
- The main job of constructor is to allocate memory for class objects.
- Constructor must be in pubic area of the class
- Constructor is automatically called when object is created.

Constructors (Contd.)

```
public class Test {

  public Test() {
  //default constructor (without parameter)
  }

  public Test(String name) {
    // This constructor has one parameter, name.
  }
}
```

Constructor name and class are same

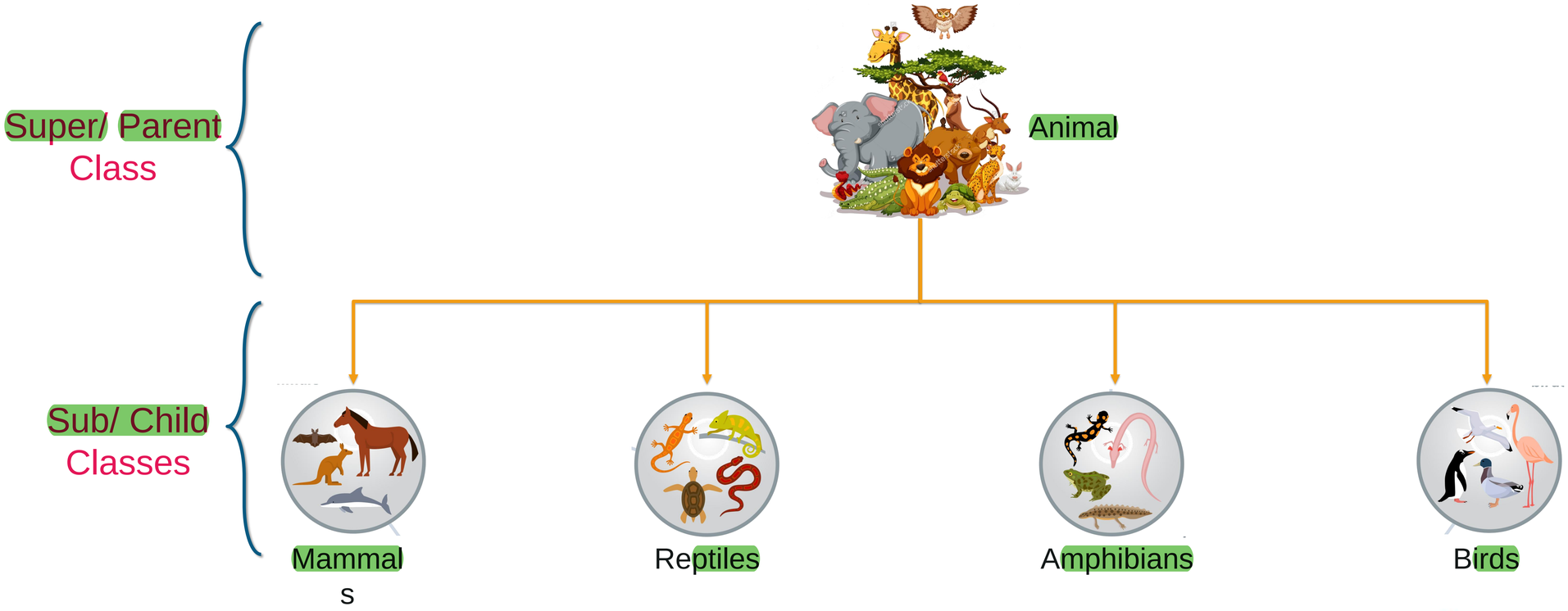
OOPs Concepts

OOPs Concepts

1. Inheritance
2. Polymorphism
3. Abstraction
4. Encapsulation

Inheritance

- Inheritance is the property of an object to acquire all the properties and behavior of its parent object
- Inheritance represents the IS-A relationship which is also known as a parent-child relationship



Inheritance

Syntax

```
class Subclass extends Superclass
{
    //methods and fields
}
```

Advantages

- Code Reusability
- Extensibility
- Overriding
- Data Hiding

Inheritance

Types Of Inheritance in Java

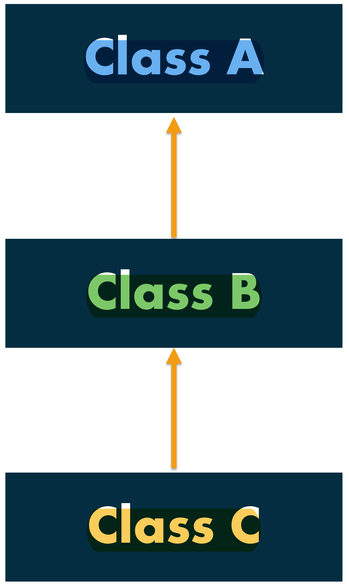
- Single
- multilevel
- hierarchical
- Multiple

Inheritance - Single

- Single level inheritance enables a derived class to inherit properties and behaviour from a single parent class

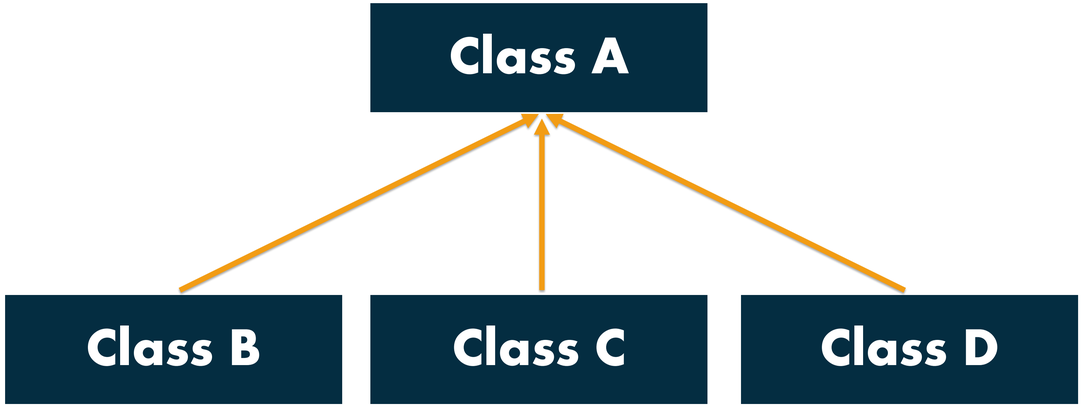
Inheritance - Multilevel

- Multi level inheritance enables a derived class to inherit properties and behaviour from a parent class which is also derived from another class



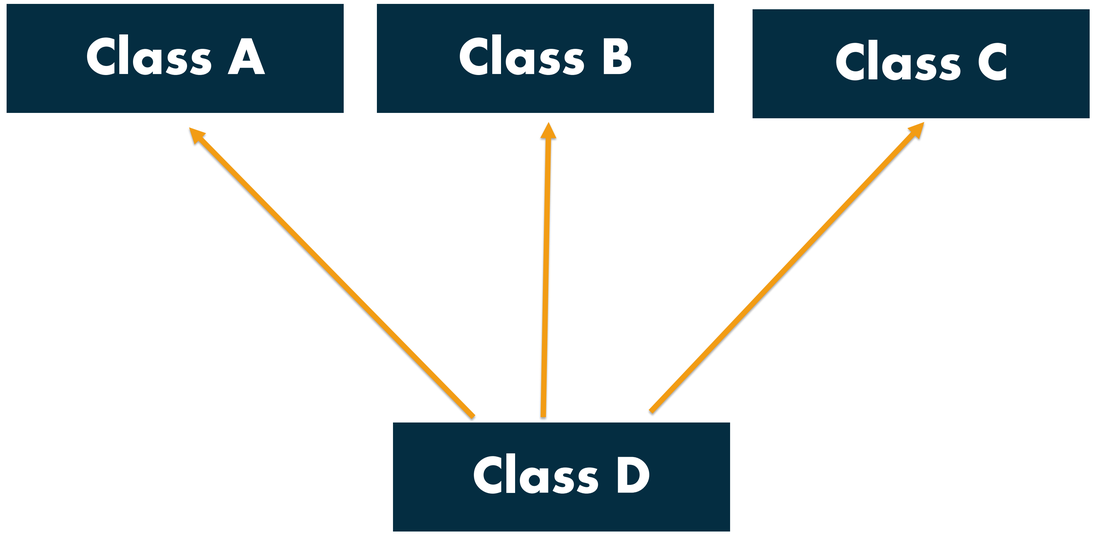
Inheritance - Hierarchical

- Hierarchical level inheritance enables more than one derived class to inherit properties and behaviour from a parent class



Inheritance - Multiple

- When We inherit from multiple classes i.e. more than 1 class.
- Multiple Inheritance is not supported by Java.



Polymorphism

- Polymorphism is the property of an object which allows it to take multiple forms

Types Of Polymorphism in Java

- Compile time
- Run time

Polymorphism – Compile Time

- Compile Time Polymorphism or Static Polymorphism is resolved during compiler time
- **Overloading** is an example of compile time polymorphism

Rules For Overloading

1. Overloaded methods must have different argument list
2. It can have different return types if argument list is different
3. It can throw different exceptions
4. It can have different access modifiers

Polymorphism – Run Time

- Run Time Polymorphism or Dynamic Polymorphism is resolved during run time
- Method Overriding is an example of run time polymorphism
- An overridden method is called through the reference variable of a superclass

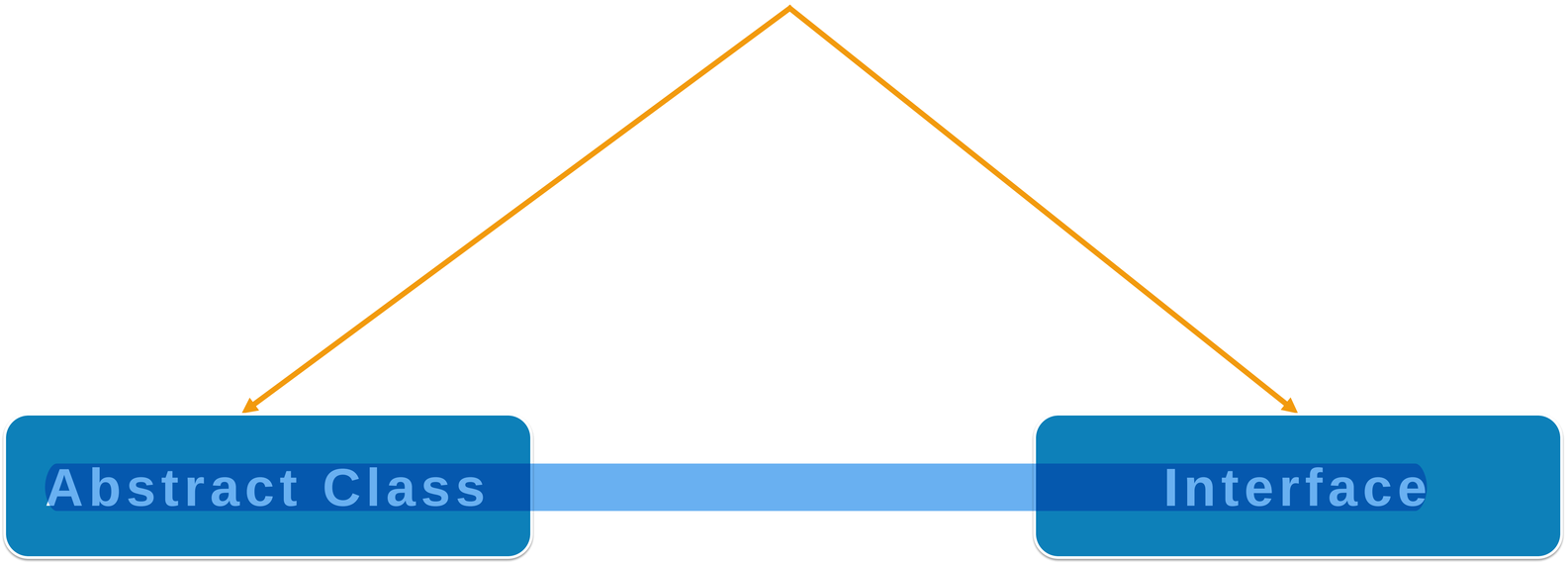
Rules For Overloading

1. Overriding method argument list must match the overridden method
2. The return type must be the same or subtype of overridden method
3. Access level cannot be more restrictive than overridden method

Abstraction

- Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to them

Ways to achieve Abstraction



Abstraction – Abstract Class

- An abstract class is a template definition to methods and variables of a class that contains one or more abstracted methods

It can provide from 0 to 100% of abstraction

Must be declared with an abstract keyword

Can have abstract and non-abstract methods

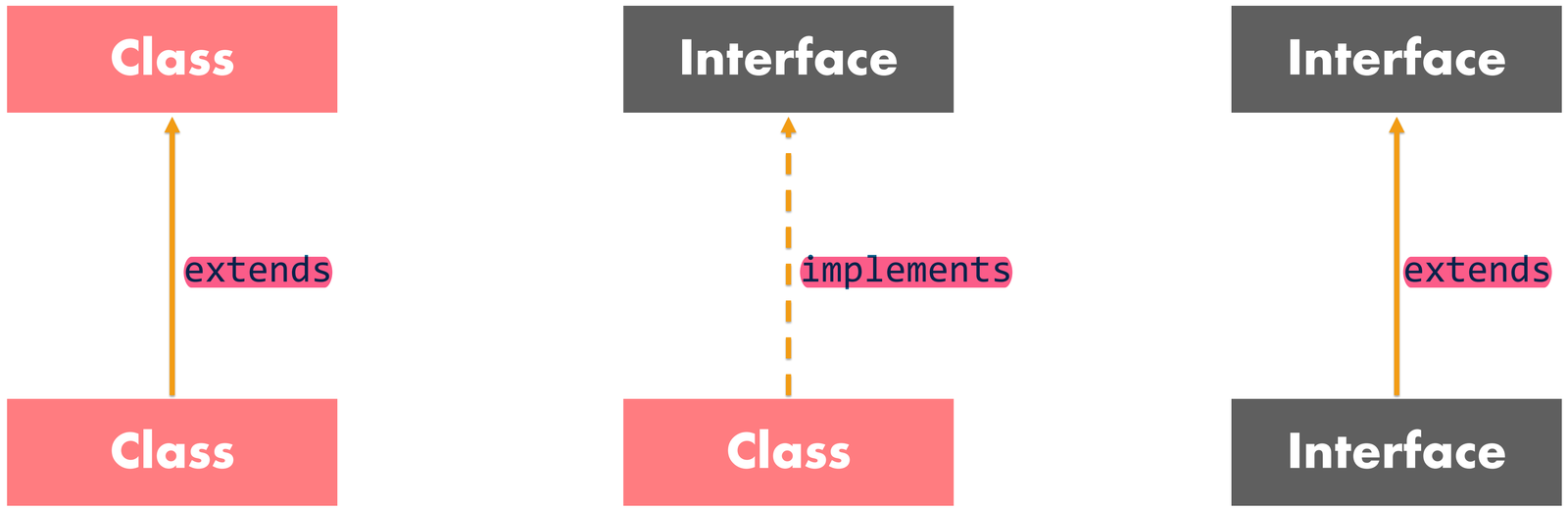
Cannot be instantiated

Can have constructors and static methods

Can have final methods

Abstraction - Interface

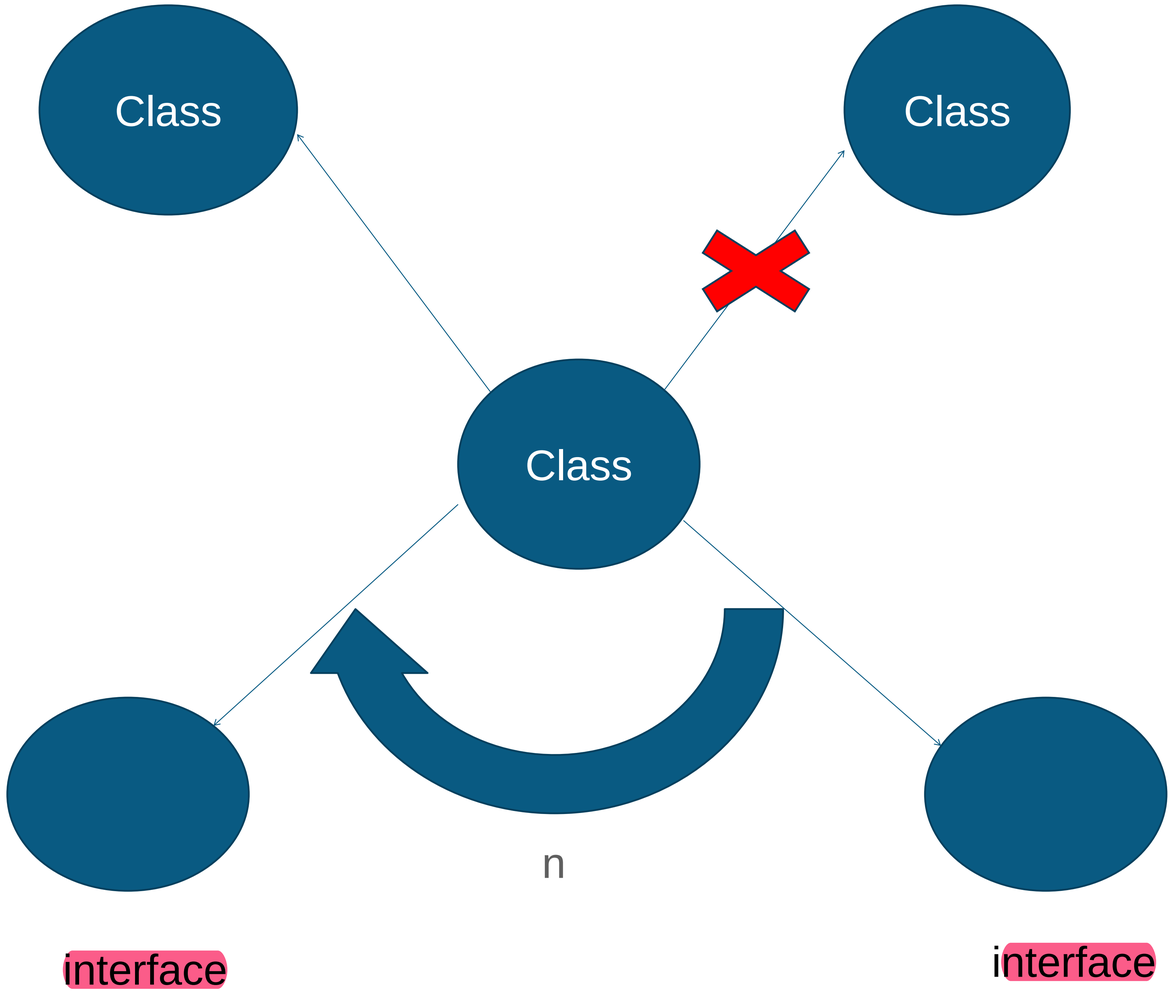
- An interface in java is a blueprint of a class which contains static constants and abstract methods
- It Enables Multiple Inheritance and helps in achieving loose coupling
- **It provides 100% of abstraction**



Why are “Interfaces” used?

- Interfaces are used to implement the expected behavior of a system/data types.

- Java does not support multiple inheritance, hence interfaces are implemented.


Class
Class
Class
n
interface
interface

Interface

- An interface has a set of method declarations.
- It does not have the method body but only method declaration.
- To use an interface in a class, “implements” keyword is used.
- In case you don’t override all the methods in the class the class has to be defined as abstract.
- An interface is same as class except class can be instantiated but interface cannot be.
- An interface can be defined by using interface keyword and the name of the interface.

INTERFACE

Implements

Class

Abstract Method1();

Abstract Method2();

Abstract Method3();

abstractMethod1()
{
_____
_____}

abstractMethod2()
{
_____
_____}

abstractMethod3()
{
_____
_____}

Where do we use interfaces ?

- Any system which needs the expected functionality, requires interfaces.
- For example , 2 basic functions of a bank is deposit and withdraw .
- **These two functions can be part of interfaces for banking applications.**

A Program on Interface

Untitled - Notepad

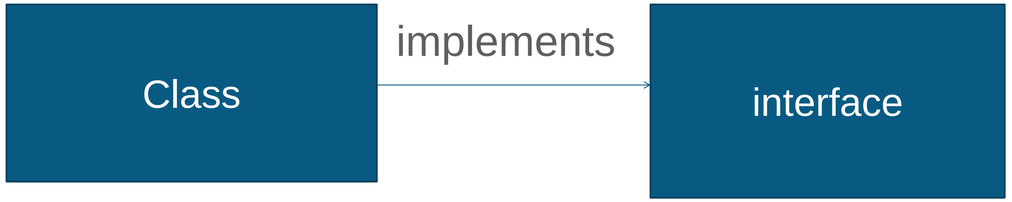
File Edit Format View Help

```
interface Area{

public double area (double length, double breadth);

}

public class rectangle implements Area {

@override

public double area(double length,double breadth) {

return length*breadth;

}

public static void main(String[] args) {
 rectangle ob= new rectangle();
System.out.println("Area of the rectangle is:" + ob.area(10, 10.5));

}
}
```

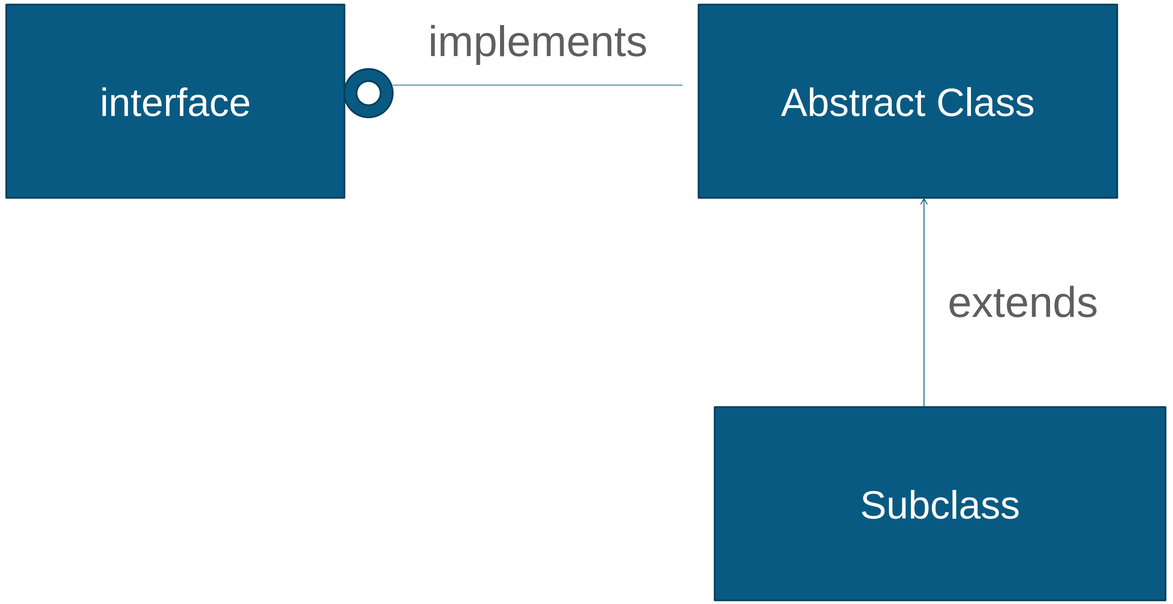
5:14 PM
3/3/2017

Interfaces (Contd.)

- Attributes can be defined in interfaces.
- These attributes can be used in the implemented class.
- The attribute values can’t be changed in the class as it acts like final variables.



Class implementing an interface



class extending an Abstract class which implements an interface.

Interfaces can be Extended

Untitled - Notepad

File Edit Format View Help

```
interface Area {
   public double area(double length, double breadth);
}

 interface Area1 extends Area {
       public double area(double radius);
}

public class rectangle implements Area1 {

   @override
   public double area(double length, double breadth) {
    return length*breadth;
}

@override
public double area(double radius) {
return 3.14*radius*radius;
}

public static void main(String[] args){
rectangle ob=new rectangle();
System.out.println("Area of the rectangle is :" +ob.area(10, 10.5));
System.out.println("Area of the circle is :" + ob.area(5.5));

}
}
```

5:41 PM
3/3/2017

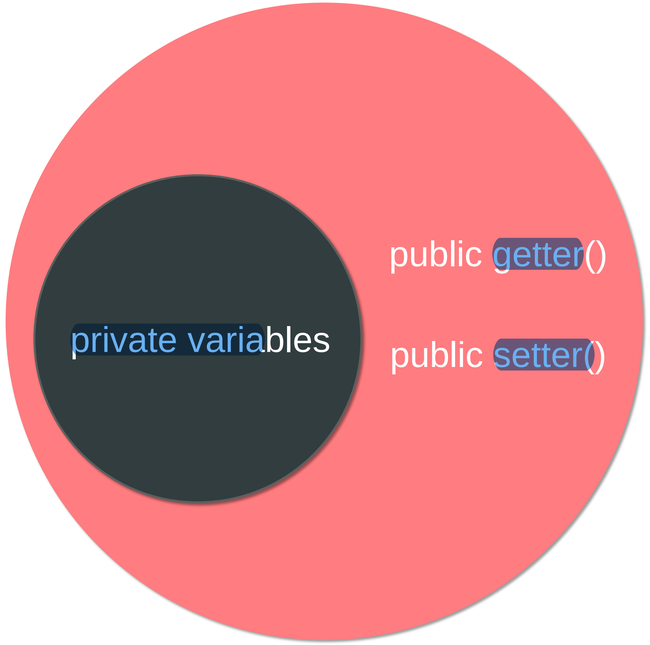
Interfaces can be Extended (Contd.)

- Like the classes can be extended, even interfaces can be extended .
- In the above program, there are two interfaces, interface 1 and interface 2.
- Interfaces 2 is extended from interface1. If a class is implementing interface2 then all the methods of interface2 and 1 should be implemented , as interface2 is extended from interface1.

Encapsulation

- Encapsulation is the mechanism of wrapping up of data and code acting on the methods together as a single unit.

- It is achieved by declaring the variables of a class as private and then providing the public setter and getter methods to modify and view the variables values.



Anonymous Objects

Anonymous Objects

- Anonymous simply means nameless. An object that have no reference is known as anonymous object.
- If you have to use an object only once, anonymous object is a good approach.
- The anonymous object is created and dies instantaneously. But, still with anonymous objects work can be extracted before it dies like calling a method using the anonymous object:
- Anonymous object instantiation:

```
new Test()
```

Anonymous Objects

```
class Test{
void fact(int  n){
  int fact=1;
  for(int i=1;i<=n;i++){
   fact=fact*i;
  }
   System.out.println("factorial is "+fact);
}
public static void main(String[] args) {
    //calling method with anonymous object
    new Test().fact(5);
  }
}
```

Final Keyword

Final Keyword

Final Variable	→	Stop value change
Final Method	→	Prevent Method Overridding
Final Class	→	Prevent Inheritance

Object Class

Object Class Methods

Root Class in Java. Below are some of important functions to be discussed:

toString()
equals()
hashCode()
getClass()
wait()
notify()
etc..

Thank You!