

## Chapter 12 :-

### Abstract Classes

def. We can define abstract class as a instantiated class (you can't declare or cheat a specific instance of it).

We can use abstract classes to make interfaces or to define abstract methods and write its no implementation in any subclass specifically.

We can use **abstract** keyword as a <sup>non</sup> access modifier for abstract classes and methods as following Syntax:

abstract  
class ← ↗ non-access modifier

public abstract class className { }

abstract  
method ← ↗ non-access modifier

public abstract int methodName();

#### notes

Abstract methods are ended by ; Because they do not have any implementation within it.

In the UML Diagrams, abstract components are italicized letters as following Syntax:

/ClassName/

### notes

Any constructor in abstract classes should be defined as protected modifier. Because it is used only by subclasses 82927010 507.2dA

### def:

You might be asking for what is the meaning of W. Concrete class is a class that has an implementation for all of its methods.

We can conclude final methods in the abstract classes.

### notes

These are Two identical Syntaxes

public abstract class <abstract> public abstract class

any abstract method ~~impl~~ is implemented by the same signature and should be preceded by @Override annotation

### notes

If a subclass of an abstract superclass does not implement all the abstract methods, It should be abstract class.

we can use abstract class as datatype

~~GeometricObject obj = new Goo~~

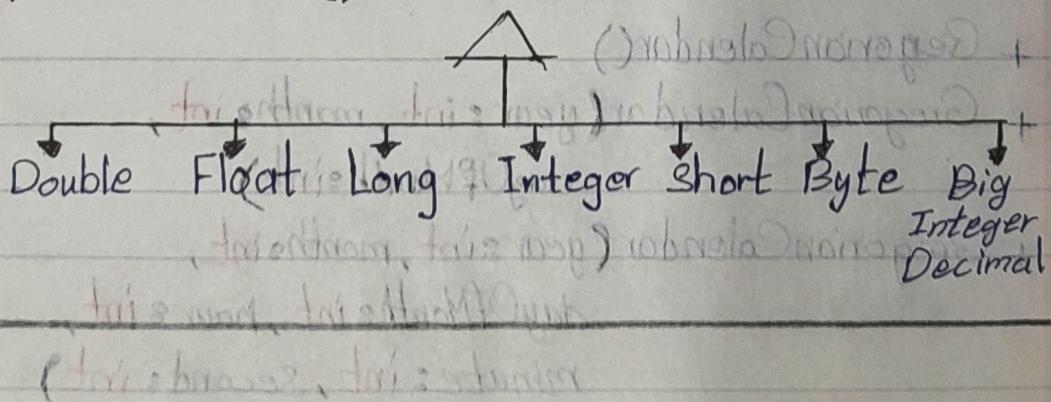
GeometricObject[] objects = new GeometricObject[10];

objects[0] = new Circle();

it's a specific case to create an instance of abstract class.

Abstract Number Class

class **java.lang.Number** {  
 + byteValue(): byte  
 + shortValue(): short  
 + intValue(): int  
 + longValue(): long }  
 + floatValue(): float  
 + doubleValue(): double }



## Abstract Calendar Class

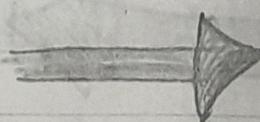
`java.util.Calendar; <abstract>`

`#Calendar()`  
+ get(field: int): int  
+ set(field: int, value: int): void  
+ set(year: int, month: int, dayOfMonth: int):  
void

`<abstract> + getActualMaximum(field: int): int  
+ add(field: int, amount: int): void  
+ getTime(): java.util.Date  
+ setTime(date: java.util.Date): void`

`java.util.GregorianCalendar; <final>`

`+ GregorianCalendar()  
+ GregorianCalendar(year: int, month: int,  
dayOfMonth: int)  
+ GregorianCalendar(year: int, month: int,  
dayOfMonth: int, hour: int,  
minute: int, second: int)`



## Constants Of Calendar

+ YEAR : int

+ DAY\_OF\_WEEK : int

+ MONTH : int

+ DAY\_OF\_MONTH : int

+ DATE : int

+ DAY\_OF\_YEAR : int

+ HOUR : int

+ WEEK\_OF\_MONTH : int

12 Hours

+ HOUR\_OF\_DAY : int

24 Hours

+ MINUTE : int

+ WEEK\_OF\_YEAR : int

+ SECOND : int

+ AM\_PM : int

## Interfaces

Def. is a class-like construct that contains only Constants and abstract methods and writing it as following Syntax

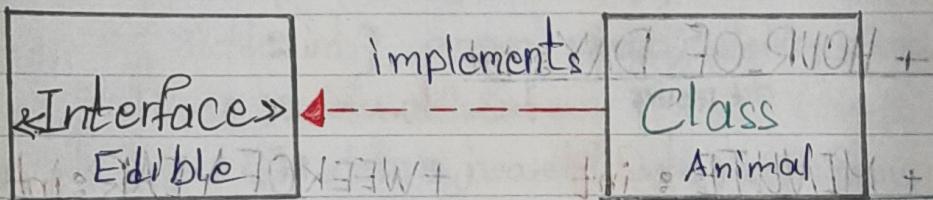
modifier interface InterfaceName { }

Notes all methods in Interface must be defined abstract

You can use the following syntax to make a class implements an interface using implements keyword

`class C1 implements InC1 { }`

The relationship between the class and Interface is known as **interface inheritance** and we can draw this relation in UML as following figure



We can use Convention Name for Interfaces that has prefix **Can** or Suffix **able**

note: Any Interface Doesn't have any Constructor

The following Two Syntax are equivalent to each other:

`public interface T { }`

`public static final int K=1; <=> int K=1;`

`public abstract void p(); <=> void p();`

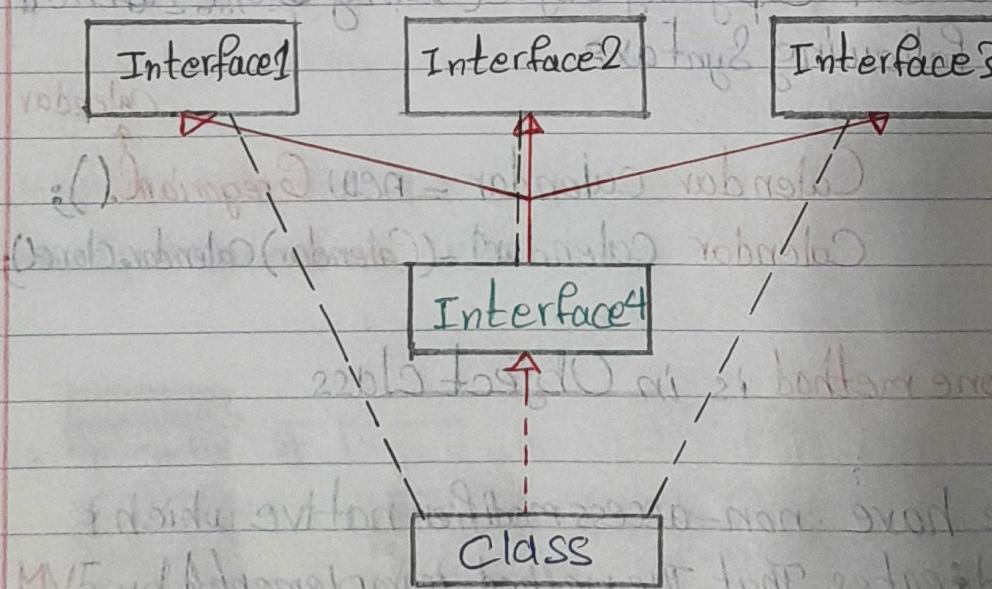
Always  
models here

## The Comparable Interface

Def: defines the `CompareTo` method for Comparing Objects and is defined as following Syntax

```
package java.lang;
public interface Comparable<E> {
    public int compareTo(E o);
}
```

java can provide multiple Inheritance Concept using Interfaces as following figure



## The Cloneable Interface

Def:

Specifies that an object can be cloned and is defined as following Syntax:

```
package java.lang;
public interface Cloneable {
    // Java.lang.Cloneable;
}
```

Def:

It is said to interface as **marker interface** when its body is empty

we can copy any object using **clone** method as following Syntax:

```
Calendar calendar = new GregorianCalendar();
Calendar calendar1 = (Calendar) calendar.clone();
```

notes

Clone method is in Object class

we have non-access modifier **native** which indicates that the method is implemented by JVM for the native platform

ex:

```
public protected native Object clone() throws CloneNotSupportedException;
```

## Interfaces VS. Abstract class

Abstract  
class

Interface

Variables: no restriction

All variables must be  
public static final

Constructors:

invoked only by  
subclasses

No Constructors

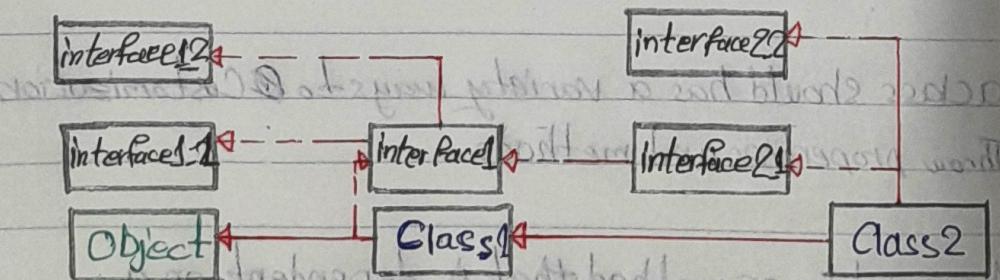
Methods:

No restrictions

All methods must be  
public abstract instance

Note:

in java interface you can make it extends from  
multiple interfaces



Note:

Interface names may be adjectives or nouns

A weak is-a relationship also known as is-kind-of relationship, indicates that an object possesses a certain property and it can be modeled using Interfaces

## Class Design Guidelines

Cohesion any class have only one identity but ~~more~~

Different responsibility

Consistency choose informative and conventional names

Encapsulation a class should use private modifier to hide data members from the user and use getters only for getting their values

Clarity a class should be readable and easy to understand from the user

Completeness a class should have a variety ways to customization through properties and methods

In Instance static A variable or method that is dependent on a specific instance variable or method. A variable that is shared by all the instances of a class should be declared static.

Inheritance

inheritance used to model is-a relationships

<sup>VS.</sup>

Aggregation: but Aggregation used to model has-a relationship

Interfaces vs.  
abstract  
classes:

interfaces used to model a weak is-a relationships  
(is-kind-of) relationship and you can achieve the  
multiple inheritance using interfaces