Chapter 13:-

Abstract Classes:

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

we can use abstract classes to make interfaces or to define abstract methods and write it's implementation in any Subclass Specifically.

we can use abstract keyword as a non access modifier for abstract classes and methods as following Syntax:

abstract class ← public abstract class ClassName { } ← non-access modifier

abstract method ← public abstract int methodName ();

notes: Abstact methods are ended by ; Because Thier do not have any implementation within it.

in the UML Diagrams, abstract components are Italicized letters as following Syntax:

ClassName

note: Any constructors in abstract classes should be defined as protected modifier. Because it used only by subclasses

def: you might be asking for what's the meaning of Concrete class is a class that has an implementation for all of its methods.

we can conclude final methods in the abstract Classes.

note: These are Two identicals Syntax:

public abstract ⟺ abstract public

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

note: 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 a datatype:

~~GeometricObject[] objects = new Geo~~

```
GeometricObject[] objects = new GeometricObject[2];

objects[0] = new Circle();
```

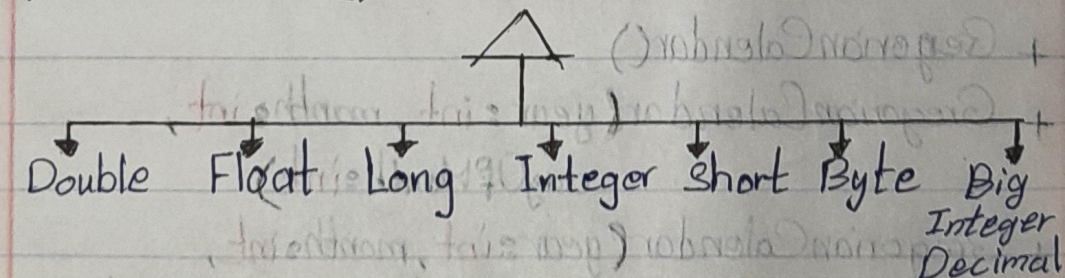
it's a specific case to create an instance of abstract class.

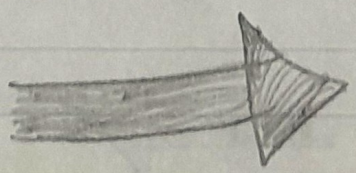
Abstract Number Class

java.lang.Number; <<abstract>>

+ byteValue(): byte
+ shortValue(): short
+ intValue(): int
+ longValue(): long
+ floatValue(): float
+ doubleValue(): double

<<abstract>>





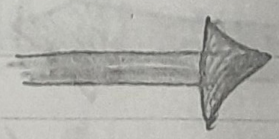
Abstract Calendar Class

java.util.Calendar; «abstract»

Calendar()
+ get(field: int): int
+ set(field: int, value: int): void
+ set(year: int, month: int, dayOfMonth: int): void
+ getActualMaximum(field: int): int
«abstract» + add(field: int, amount: int): void
+ getTime(): java.util.Date
+ setTime(date: java.util.Date): void

java.util.GregorianCalendar;

+ 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
+MONTH : int
+DATE : int
+HOUR : int
12 Hours
+HOUR_OF_DAY : int
24 Hours
+MINUTE : int
+SECOND : int

+DAY_OF_WEEK : int
+DAY_OF_MONTH : int
+DAY_OF_YEAR : int
+WEEK_OF_MONTH : int
+WEEKOF_YEAR : int
+AM_PM : int
0 1

Interfaces:

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

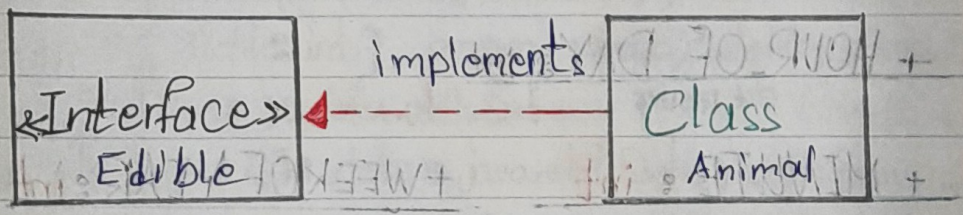
```
modifier interface InterfaceName {}
```

Notes: all methods in Interface ~~should~~ must be defined abstact

you can use the following Syntax to make a Class implements an interface using implements keyword

```
public 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 Constractor

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();
}
```

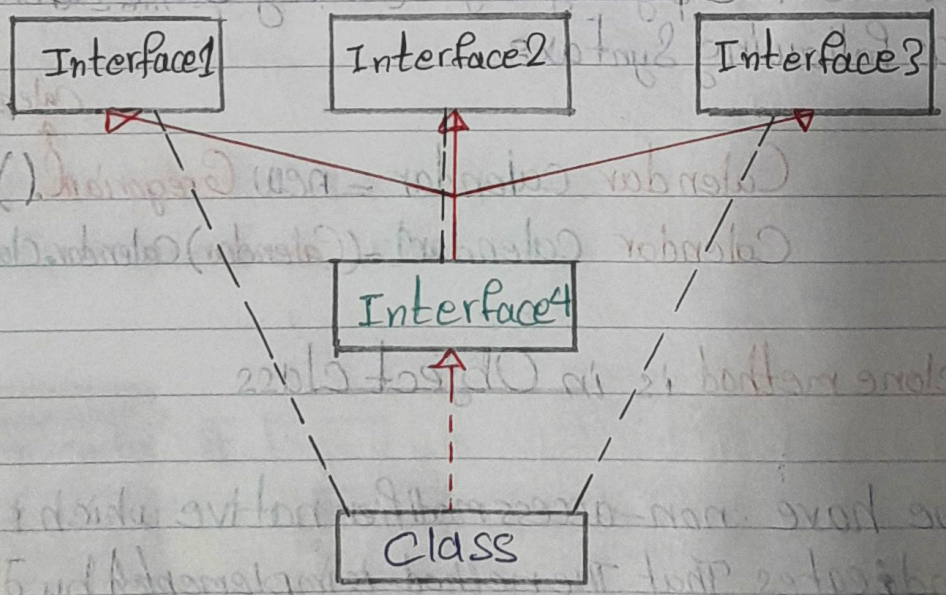
Always modefiers

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.lang.Comparable;
```

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 Clonable {
} //: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 Gregorian C.();
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

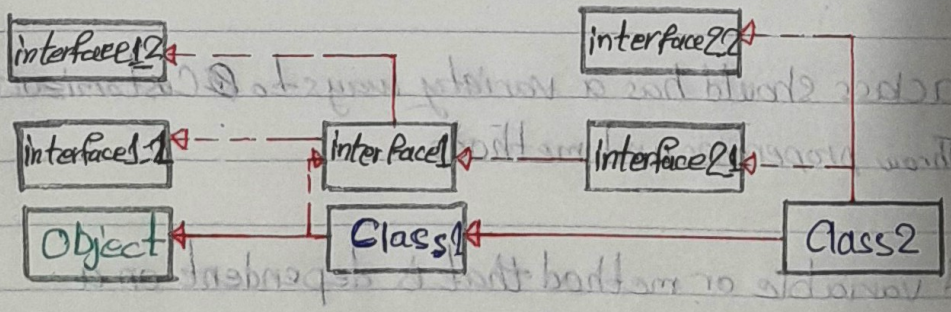
ex:

```
protected native Object clone() throw 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 methods

note: in java interface you can makes it extends from multiple interfaces subinterface



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 ~~there~~ Different responsiblity

Consistency Choose informative and convention 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 by readable and easy to understand from the user

Completeness aclass should has a variaty ways to Customization Throw properties and methods

Instanse Vs static A variable or method that is dependent on a specific instane variable or method. A variable that is shared by all the instances of a class should be Declared static.

Inheritance vs. Aggregation: inhertance used to model is-a relationships 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