COURSE ERA : JAVA Programming

Dec 6

Overview

1) why classes & objects.
2) creating classes, objects
3) Member variable / Methods / Constructors

Classes → a data type → think of it like a factory

Objects → a data of class data type → an object coming out of factory

Naming :

Java file name ⇒ should be same as public class name.

1) Program to Represent current location in map:

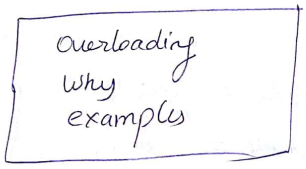
```
public class SimpleLocation
{
    Public double latitude;
    public double longitude;

    public SimpleLocation (double lat,
                           double long)
    {
     // Constructor
        this. latitude = lat;
        this. longitude = long;
    }
    Public distance ( SimpleLocation Other)
    {
     // Body
    }
}
```

Member Variables / Instance Variables of Class

- In prev_example
  latitude & longitude are mem_variables
  The ones which are NOT in any methods

lec 2

overview



Two Constructors in one Class

(1) Example

```
class X_class
{
    Public x_class () {
    } // Constructor 1
    Public x_class (double l, double m){
    } // Constructor 2
```

- Here Constructor method is overloaded
- Overloading other function in a class

(2) Example

```
public class Simplelocation
{
    ----
    public double distance (Simplelocation other)
    {
        // Body
    }
    Public double distance(double l, double m)
    {
        // Body
    }
    ----
}
```

(3) What we cannot do ??

- cannot have two methods with same parameter list & just changing return type. See ⟹

```
class sample                                    NOT ALLOWED
{
  public double distance (int a, int b)
  {
  }
  Public int distance (int c, int d)
  {
  }
```

lec 3

overview

Keywords :
public / private / protected

- Public data members & Methods can be accessed from another class, using the first class object.

- Private

```
public class Simplelocation
{
  private latitude;   } Can be accessed
  private longitude;  } ONLY within this
                        class
  public Simplelocation ()
  {
    this.latitude = 10; → allowed
  }
}
Public class location_tester → external class
{
  Public static void main ()
  {
    Simplelocation obj1 = new Simplelocation();
    obj1.latitude = 20; ⇒ NOT ALLOWED
  }
}
```

RULE OF THUMB

- Mem_Variables : declare private.
- Methods : either public / private.

Accessing private mem_variables outside

→ use the public methods of source class to get the value of variable.

getter functions

```
Public class SimpleLocation
{
  Private double l;
  public double getter ()        // GETTER
  {
    return l;
  }
}
```

Similarly SETTER method, we can use in the desired class, to change the variable (private).

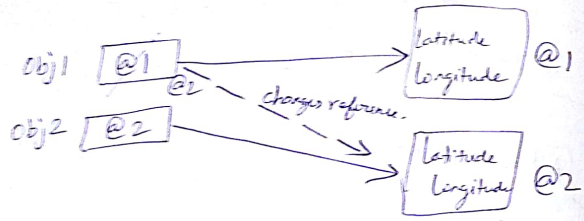
Dec 11

Memory modes

Objects :

```
Public class SimpleLocation
{
  ~~~
  ~~~
}
Public class tester
{
public static void main (String[] args)
{
    SimpleLocation obj1 = new SimpleLocation (5, 10);
    SimpleLocation obj2 = new SimpleLocation (15, 20);
    obj1 = obj2 ; //changes reference of Mem. location to obj2's location.
    obj1. Latitude = 25;
    System.out.println (obj2.latitude);
    OUTPUT ⇒ 25 //
```

obj1 = obj2 ; [What happens ?? Shown below dotted lines]



Java Datatypes:

Byte
Short
int
Long
Float
Double

Console input:

"Scanner class"

```
import java.util.Scanner
```

(1) First create an object for Scanner class.
(2) Use the methods of "Scanner class", to get any kind of input.

```
Scanner input_vig = new Scanner(System.in)
int a = input_vig.nextInt(); // Prompts for i/p
```

Similarly;
nextDouble()
nextFloat()
next() - Reads str, till whitespace.
nextLine() - till a Return Key.

Constant variables:

```
final int PI_VAL = 3.14;
```

final → Keyword; int → data type; PI_VAL → upper case; 3.14 → Value.

integers Max_limit

int → 4 Bytes (-2^{31} to $2^{31}-1$)
(-2147483648 to 2147483647)
if you do, int i = 2147483647 + 1
↳ stores -2147483648 (overflow)

Increment / decrement

i=1;
j=i++; → j becomes 1, i becomes 2

i=1;
j=++i; → j becomes 2, i becomes 2

Strings:

String s1 = "Vignesh"

To read strings,

```
Scanner input = new Scanner(System.in)
String s1 = input.next();
String s2 = input.next();
```

output ⇒ s1 & s2

Console : > "Hi there"
(Hi → s1, there → s2)

To get a character:

input.next().charAt(0)

(input.next() → This gets String; charAt → This gets Character; 0 → index)

String s1 = input.nextLine() → entire line

2

String to int / int to s

String s = "123";

(1) use the Integer class method parseInt

int a = Integer.parseInt(s);

||ly for double,

double d = Double.parseDouble(s);

Random Numbers

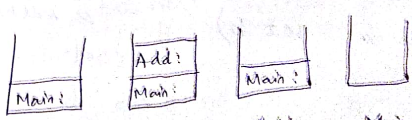
→ Turn 5 pages for more info

Monte-carlo random numbers ⇒ Math.random()

double a = Math.random();

Call stack : what is it ??

A stack of methods currently executing



Main executing | Add came in, executing | Add finished executing | Main too finished. Stack empty.

Function overloading :

GOOD way

```
int max (int a , int b)
{
 ---
}
double max (double a , double b)
{
---
}
int max (int a, int b, int c)
{
---
}
```

Ambiguous way : (Compile error)

```
int max (int a, double b)
{
---
}
int max (double a, int b)
{
---
}
```

} :=> Compiler cannot choose a method.

Single dimensional Arrays :

How to declare & Use ???

(1) Declare !

```
int [] myArray; (preferred)
        or
int myArray[];
```

NOTE :

- It does not allocate any memory
- It just created a storage location of Reference to an Array.
- So, the myArray is not pointing to any memory location, just NULL.

(2) Create Array :

```
myArray = new int[10];
```

NOTE :

- This statement created array elements, 10 numbers
- And assigned the same to reference Variable myArray.

(3) Initialization :

- After declaring & creating we can initialize individually like this;

```
myArray[0] = 0;   → initializing individual elements.
myArray[9] = 9;
```

- Another way !

declaration with initialization:

```
int[] myArray = {0, 1, 2 .... 9}
```

↓ This statement declares Creates & initializes array

But, this is wrong

```
int[] myArray;
myArray = {0, 1, 2 ..... 9}
```

copying Arrays :

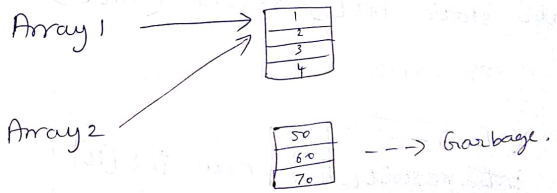
Before

Array 1 → 1 2 3 4

Array 2 → 50 60 70

```
Array 2 = Array 1;
```

After :



- Does NOT copy contents !
- changes reference.

Array passing in arguments :

```
function = max (array_name);
---
void function = max( int[] some_name) {---
```

- Array Reference is [illegible]
- So, modifying values in function, modify actual array values.

→ like C++, like operating directly on address of variables.

Returning & Receiving Array

```
Public static int[] reverse (int[] name)
{
    ------
    int result[] = new int[10];
    ------
    return result; ---> returning
}

Public void main() {
    Array declare & define myArray[] //
    int[] list2 = reverse (myArray);
}
```

<---> Storing

Variable length Argument parameter

(1) printMax (double a, double b) { - - - }

↳ can receive only two arguments.

But what if we need variable length arguments ??

Use this :

printMax (double... a) { - - - }

↙

The arguments will be stored from a[0] .... any # of arguments.

a.length → for len.

---

java.util.Arrays → Arrays class

- Searching / Sorting / comparing / filling arrays
- java.util.Arrays.sort (array_name)
  ↳ sort whole array.
- java.util.Arrays.sort (array_Name, 0, 3)
  ↳ Sort only from 0 to 2 elements

- java.util.Arrays.binarySearch(list, 1);
- java.util.Arrays.binarySearch(list, 'a');

Compare: (two arrays)

- java.util.Arrays.equal(list1, list2) → o/p True/false

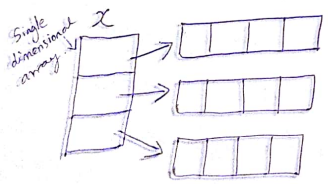
Fill: (Fill an array with Same Value)

- java.util.Arrays.fill(list1, 5)
  - list1 → array name
  - 5 → fill with 5

Two-dimensional Arrays:

(1) declare `int [][] x;`

(2) Create `x = new int [3][4]`

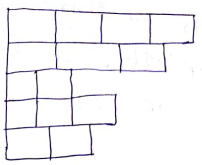


x.length ⇒ 3

x[0].length ⇒ 4
x[1].length ⇒ 4
x[2].length ⇒ 4

"A two dimensional array is a single dimensional array, with each element being an array itself"

Ragged arrays:



→ different sized arrays.

(1) declare: `int x[5][];`
or
`int[5][] x;`

(2) Create:

```
x[0] = new int[4];
x[1] = new int[3];
:
x[4] = new int[2];
```

Constructors:

- NO return type needed
- Just the name - same as class Name

```
class_name() → Constructor.
{
  ===
}
```

Objects – Garbage Collection :

- if an object of class is no longer needed assign it to NULL explicitly, garbage collection routine takes care of it.

Random Numbers :

```
Random random1 = new Random (3);
```

new object

This is "seed" value

- if left Blanks, current time used as seed.
- if any specific value mentioned, generate random sequence for that seed.
- for Any two objects with same seed value, the random seq will be identical.

```
Random random2 = new Random(3);
```

Same seed So ||| sequence

"STATIC" - what is it ??

- The static keyword indicates that, the particular variable is a static / class variable. which means, the variable is shared among all objects of same class.

In other words, a single mem. location is used for all objects.

"Instance" variables :

- If static keyword not used, its an instance variable.

- Its just an instance of a class, so every object will have different instances

Static	instance
• static variables & methods can be accessed Via class name, before even creating an object for the class.	• Instance variables & methods can be accessed only after creating an instance of the class, i.e the objects.
eg:	eg:
Classname. Variable_name = 5;	Circle c1 = new Circle();
Classname. getArea ();	c1. variable_name = 5;
	c1. getArea ();

- Static variables can be used anywhere, : static or instance methods

- But, instance variables can be used in instance methods only.
- But if you create an instance, you can use anywhere (create obj & access through obj)

Static ? or Instance ? Declaration

→ Imagine a Circle class
the radius variable has to be an instance (because different Circle objects, diff radius)

→ The Math functions : are static methods

Visibility Modifiers :

(1) Accessibility of classes :

→ public classes are accessible among different packages.
(or)
public classes are accessible everywhere.

→ Private classes are accessible within that particular package alone.

(2) different Modifiers :

public / default / private / protected.

- private members to a class (data & methods) are accessible only within that class.
- To access a private data member of another class, we can use public methods of that desired class.

[Remember getter & setter ??]

- "default" type members of a class, (means no keyword provided) are accessible within that package, among different classes.

Prevent obj creation

→ like, for static members class (Math)

: To prevent the user from creating an object for a class, make the constructor private.

example : for Math class, constructor is :

```
private Math () {
}
```

Intuitive Questions :

① Can you directly access static data members & methods in a non-static (instance) method ??

② Can you access the instance Varibles or instance methods from an static method ?? If yes, what modification is needed.

③ How to prevent a user form creating an object, bcoz the class has static [methods & variables] & no need to create an object.

④ How objects are initialized?

⑤ when does the Constructor of a Class get invoked ?? which Keyword exactly ??

⑥ How array of objects declared, created and initialized ???

Strings :

• In C, C++ strings are nothing but array of characters.

But in Java, each string is an object of String class. This string class has many constructors & Methods, to manipulate strings.

Construct a String / Create a string :

(1) using direct string literals → "Hello world"

→ String myString = new String ("Hello world");

or

→ String myString = "Hello world";

or

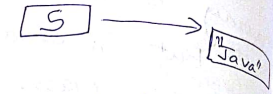
→ char[] charArray = {'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd'};

String myString = new String (charArray);

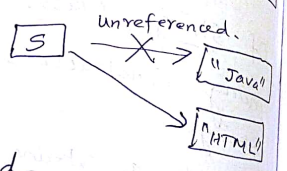
Immutable Strings :

→ Contents cannot be changed.

- String s = "Java";



- s = "HTML";



This statement created new object for String class with contents as "HTML" and assigned the same to reference 's'. But the old object is still out there.

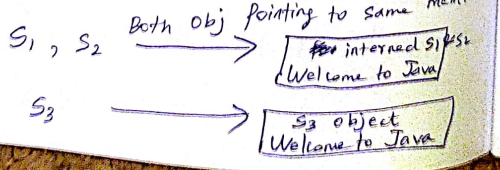
Interned strings :

→ JVM technique to save space.

→ when strings are created like below,

String s1 = "Welcome to Java";
String s2 = "Welcome to Java";
String s3 = new String("Welcome to Java");

Memory Map :



Some methods : (Java.lang.String)

equals
equalsIgnoreCase
compareTo
compareToIgnoreCase
regionMatches
startsWith
endsWith

---

myString.length
myString.charAt(index)
s3 = s1 + s2; (concat)

Substring(~, ~)

---

toLower
toUpper
trim
replace → replace characters
replaceFirst → replace 1st matching substring
replaceAll → replace all matching substring
split → returns array of str, split by delimiter.

indexOf
lastIndexOf

Prog. Exercise :
1) Count each letters in a string.
eg:- > Enter string :
Hello
O/p :
H - 1
e - 1
L - 2
O - 1

Refer Book Daniel Liang Pg : 315

StringBuilder class → for handling strings more flexible
Suitable for single task access.

StringBuffer class → Suitable for Concurrent tasks - accesses on string.

FILES in Java.

Why? to store data permanently in a disk/CD.

Absolute file name :
Windows : C:\book\welcome.java.
Linux : /home/liang/book/welcome.java

java.io.File class : to get basic attributes of a file.

Methods :

Java.io.File file = new java.io.File("Dir/vig.txt")

file.exists()
file.length()
file.canRead()
file.canWrite()
file.isDirectory()
file.isFile()
file.isAbsolute()
file.isHidden()
file.getAbsolutePath()
file.lastModified()

This File class does not have the Read /write method

java.io.PrintWriter class : to write data to file

java.io.PrintWriter output = new java.io.PrintWriter(file);
obj of File class

invoking constructor of printwriter will create a new file, if file is not alread exist. else discard the data of existing file.

```
output.print("Vigneshwar padmonaban");
output.close();
```

---

java.util.Scanner : to Read Data

```
java.io.File file = new java.io.File("scores.txt");
Scanner input = new Scanner(file);
while (input.hasNext())
{
    String fName = input.next();
    String mName = input.next();
    String lName = input.next();
    int score = input.next();
}
input.close();
```

Thinking in OOP way :

Immutable objects :

- Usually we create objects, whose content will be changed at later points.
- But, if an object's contents are not changed and remains fixed, its an immutable object.

Conditions to satisfy for immutable object

1) All data fields are private
2) No mutator methods - other words, no setter functions, which can modify the data fields.
3) no ~~data~~ accessor method (or Getter method) who helps to get value of a variable, which changes. Like a method which returns date field, time etc...

Scope of variables :

class variable Vs local variable :

```
Class Test
{
  Private int x; x = 0;
  Public void display(){
      int x = 1;
      Print x;
  }
}
```

→ "Here, two variables of same name. So, the local variable x, takes precedence. So value printed is 1."

"this" reference:

- lets consider an instance method

```
{ public void setI(int a)
  {
    this.i = a;
  }
}
public TesterClass
{
  public static void main(String[] args)
  {
  // Creating object for the above class → O1.
  // call the SetI method:
  O1.setI(10);
```

↓ Object O1 is invoking SetI.

the "this" pointer, refers object O1.

} }

Here, 'i' is a private variable in a class, also non-static

Inheritance:

Base class / Parent class / Super class

↑

Derived class / child class / Sub class

example:

```
Public class GeometricObject1 {
~~
}
Public class Rectangle extends GeometricObject1 {
~~
}
```

NOTE:

1) Private members cannot be inherited, but yes accessed through public accessors / mutators.

2) Java allows to extend only one class at a time. ONLY ONE PARENT

But, multiple inheritance achieved through INTERFACES.

Super Keyword :

- While inheriting from parent class, do we (Child class) get the Constructors of parent too??

Answer: No. we don't inherit Constructors, But we can invoke using "Super" Keyword.

- we can invoke superclass Constructors from Subclass Constructors, using "Super"
  → with "Super" being the first line in Subclass Constructors.

```
public class Circle{
    public Circle()
    {
    ~~
    }
}~~~
```

```
Public class CircleChild extends Circle {
    public Circle child ()
    {
    super (); → this invokes no-arg Constructor
    ~~          in super class.
    }
}
```

- if parameters present
  super (parm1, parm2);
- We can also invoke methods of Superclass from "Super" Keyword.

Constructor chaining :

- Even if we don't include "Super" Keyword in the derived class, super () will be implicitly invoked Base class no-arg Constructor

NOTE: so its good to have no-arg Constructor in any class to avoid error. Because, if child class did not include "Super" keyword explicitly, super() invoked.

idea is, before the instance of child class is Constructed, Base class instance need to be Constructed, So invoking their Constructors.

Refer Daniel Liang Book, Pg 381 for example.

"Super" keyword to invoke Superclass methods

- Apart from calling super class constructors, superclass methods can be invoked by,

  super.methodname(parms);

But, why do we need ?? As the method is already inheritted, can be invoked normally methodname(parms);

purpose example → while overriding superclass methods.

- Say there is a method in superclass named ⇒ toString() { ~~~ }

In derived class, the toString() is overridden as

```
toString() {
    return super.toString() + " is the answer"
}
```

↓

we just appended a string to Super class methods response.

overloading Vs overriding

Overloading :

- method name same
- different signatures (or) parameters

Overriding :

- method name same
- same signature or parameters.

The Object Class :

- Every class in java descended from the Object class.
- eg: one method of Object class Is → toString()

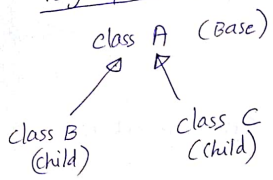
```
classname o1 = new classname();
```

↓ object

```
System.out.print(o1.toString())
```

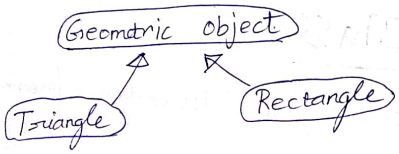
↓ details about the object o1 is printed.

Polymorphism :



- Every instance of a child is an instance of parent too !!

But, its not true for vice-versa.



Therefore, you can send the instance of child class to parent class

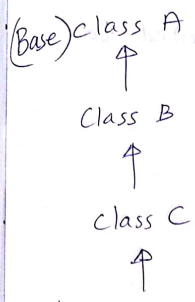
⇒ Class B o1 = new class B( ); → Sent
display(o1);

public static void display (class A b1) → recvd
{
}

↳ receiving the obj of class B with class A (super class)

Dynamic Binding :

dynamically binding the call to a method. Lets say there are many classes with parent-child relations & many has overridden methods, which method to call ??



- Lets say a method named : toString() is invoked, using class D object.
- See if class D has defined toString() no? check its parent ⇒ class C. go till class A. Still cannot see any method named toString() ?? check its parent Object

java.lang.Object class is parent of all classes isn't it?

Type casting objects & "instanceof" Keyword:-

- Object o = new Circle();

  ↓

  Here object o, implicitly type casted to Circle class.

- Explicit type casting:

Circle o = (Circle) o;

---

instanceof: → Keyword to check if an object is an instance of a class.

```
display (Object o)
{
  if (o instanceof Circle)
  {
    ~~
  }
}
```

Prevent inheritance & Overriding

① To prevent a class to be extended by another class, "final" Keyword.

```
public final class classname {
  ~~
}
```

② Or, to prevent a method from overridden by derived classes, "final" Keyword.

```
Public final void m () {
  ~~
}
```

Exception Handling :

```
try
{
  ~~
}
catch ()
{
  ~~
}
```

in detail

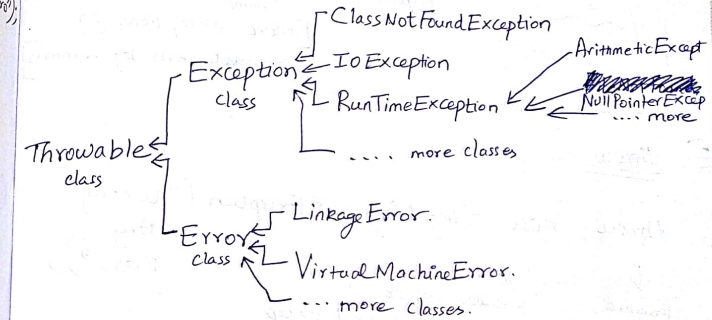
```
try
{
  ~~ condition
  throw new ArithmeticException("Divisor cannot be zero");
  ~~
}
catch (ArithmeticException ex)
{
  //do something Stop or Continue.
}
```

FileNotFoundException:

```
try
{
  Scanner input = new Scanner(file);
  ~~
}
Catch (FileNotFoundException ex)
{
  ~~
}
```

InputMismatchException → giving character input for integer prompt

Throwable class : The Root class for all exceptions & errors in java.



- **User defined Exceptions** :

  we can extend Exception class to Create our own.

Handling Exception

- declare exception
- Throw exception
- Catch exception.

① declare : No need for Run time exceptions or errors.

for others, need to declare exception like:

```
methodname throws IOException () {
}
```

throws → keyword; IOException → exception name.

[More exceptions ?? Separate em by commas]

② throw :

```
throw new IllegalArgumentException ("Wrong Arg Boss");
```

throw → keyword; "creates an object of the specified exception Class".

③ Catch :

```
method1 {
  try {
    invoke method2;
  }
  catch(){
  }
```

```
method2 {
  try {
    invoke method3;
  }
  catch (){
  }
```

In method3, if an exception occurs, an appropriate catch statement is searched in method3, if not found then searched in method2, then the search will come to the main method where, appropriate catch() is searched. If nowhere found, program executions STOPS. printed error.

NOTE : Order of catch () blocks can cause error !!!

```
Catch (Exception ex) {
}
Catch (RunTimeException ex) {
}
```

This is error coz, block of superclass appears above Subclass. To correct, reverse order.

NOTE : After "throw" statement in try { ... } block, execution transferred to catch block, then finally { ... }. Then statements after finally block.

→ cannot have statements after throw statement. [if throw not exec, due to condition, we can have other Stmts.]

```
try {
  if (condition) {
    throw ~~~
  }
  ~~~ //other statements if no throw
  ~~~ //
}
catch (~~~~) {
  ~~~
}
finally {
  ~~~ // These stmts exec, no matter what either excep or not.
}
```

Rethrow

- catch() method, which catches the exception can do some work, then rethrow the same exception!

```
catch (~~ex) {
  // print warnings
  throw ex;
}
```

Chained Exception

- throwing new exception along with a already captured exception.

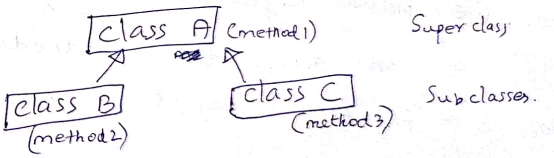
```
catch (Exception ex) {
  ~~~
  ~~~
  throw new Exception ("New info", ex);
}
```

ex → old exception

Abstract classes

- Its a sort of declaring prototypes of methods; of sub-class in Super-class.

say,



- From the instance of class A, to use the methods of class B and class C we declare abstract of those methods in Class A.

- Also include Keyword in class

```
public abbtract class classA {
    method1 () {
    }
    public abstract void method2 ();
    public abstract void method3 ();
}
```

NOTE: if we are including abstract methods in class, make class abstract too. (i.e) include keyword ("abstract")

NOW, we can access these methods of class B & C from object of class A as follows:

```
classA obj1 = new classB ();   --> Note the different objects.
classA obj2 = new classC ();
a=obj1.method2();
b=obj2.method3();
```

Abstract classes – Points to Note :

① Abstract methods can be contained only in abstract classes.

② An Abstract superclass can have either abstract or non-abstract sub-classes based on – if all methods of superclass implemented in sub-class, it can be non-abstract otherwise abstract.

③ Abstract classes are non-static.

④ Abstract classes should have protected constructors.

Meaning → Cannot create an instance of an Abstract class from anywhere, except from sub-class constructors.

⑤ we can define an abstract class without having any abstract methods.

⑥ A subclass can be abstract, even if the superclass is concrete (or non-abstract)

⑦ In subclass, overriding a method declared as abstract in its superclass, is allowed. [provided subclass declared abstract]

(8) can we create array of objects of abstract class type ??

Yes

from our previous example,

```
classA[] objects = new ClassA[10]
```

then,

initialization part

```
objects[0] = new classB();
objects[1] = new classC();
```

→ Here, the constructors of class B is creating instance of class A & class B.

Interface :

- We can implement any number of interfaces in a class, But can extend only one class.
- interface is similar to class inheritance.

Syntax :

```
public interface Interface_name {
    public final static int NUMBER;
    int ID; // ---> implicitly assumed public final static
    public abstract void p();
    void add();
}
```

- In Interface, (irrespective of keyword given)
  all data fields – public final static
  all methods – public abstract.
- NO Constructors in Interface.

Usage

```
class Class_Name extends Some_class implements
                                    Interface_name {
    // The abstract methods of interface
    // needs to be defined here in this class
    // or any sub-classes of it.
    public void p() {
        ~~~
    }

    // To access the data fields of
    // Interface, same as a static class
int a = Interface_name.NUMBER
}
```

Diff b/w Abstract Classes & Interfaces

- Refer page 475, Daniel Liang 8th ed.
- **using Abstract classes :**

class A .....→ abstract superclass

Class B, class C .....→ sub classes extending the abstract superclass.

Here the abstract methods of Class A defined inside class B & class C.

But, the object of class A, when it is used to invoke a abstract method, its dynamically decided to invoke which method in which class.

```
classA obj = new classB
obj.method();  ---> invokes classB's method
```

- So, using Abstract classes, only those subclasses are coming under scope.

using Interface :

Any class can come under scope.

no-need of any parent/child relationship.

```
interface Interface_Name {
    // method decl. here.
}
class classA implements Interface_Name {
    //method definitions.
}
class classB implements Interface_Name {
    //method def.
}
```

↳ As you can see, classA & classB are totally unrelated classes. Yet they can implement a Command Interface.

Conclusion : Interfaces offers more flexibility

Java import Vs extend (inheritance)

- `import` is just to avoid writing full path everytime.

for example:

If we don't import java.util.Scanner we can still use Scanner class by

⇒ java.util.Scanner input = new java.util.Scanner(System.in);

(java.util.Scanner: Full path)

`extend`:

This is inheritance. Inherit the members like data members & methods, which we can access with Child class object. (of course except the private mem of Baseclass)