Lesson Outcomes

At the end of the chapter, students should be able to:

- ✓ Understand the concept of object and class in OOP
- ✓ Define field and method in class
- ✓ Understand and use access modifier
- ✓ Understand and implement inheritance, polymorphism
- ✓ Understand how one class relate to one class
- ✓ Write program using OOP languages

A. INTRODUCTION TO OBJECT-ORIENTED PARADIGM

- OOP focuses on data structures with functionality or processing capability to those structures.
- Data structure definition and its defined processes are package together in some syntactic structure, in which the structural definition and process implementation are hidden from the program unit that uses it, which are called clients.
- The world around us is made up of objects such as people, automobiles, buildings, streets, adding machines, papers, and so forth – each of these objects has the ability to perform certain actions, and each of these actions has some effect on some of the other objects in the world.
- OOP is a programming methodology that views a program as similarly consisting of objects that interact with each other by means of action.

Terminology of Object, Class and Methods

OOP has its own specialized terminologies which are object, class and methods.

- Object:
  - Represent entity that has a state, exhibits some well-defined behaviour and has a unique identity.
  - Is a self-contained entity which has its own private collection of properties (i.e. data) and methods (i.e. operation) that encapsulate functionality into a reusable and dynamically loaded structure.
- Class:
  - Represent a set or collection of objects that share a common structure and a common behaviour.
- Field:
  - Attribute of the class object.
  - Two types:
    i. Primitive, i.e. int, long, double etc.
    ii. Reference data types (ADT), i.e. String
- Method:
  - Represent the action of object
  - Two types of methods:
    i. User defined methods: constructor (default, normal, copy), destructor, storer/mutator, retriever/accessor, processor and printer.
    ii. Built-in methods

Fundamentals characteristics of OOP are:

- Abstraction / Information Hiding:
  - Separate the description of how to use a class from the implementation details, such as how the class methods are defined.
  - Information hiding avoids information overloading.
- Encapsulation
  - Grouping software into a unit in such a way that it is easy to use because there is a well-defined simple interface.
  - The data and the actions are combined into a single item and the details of the implementation are hidden, i.e. a class object.
  - A programmer who uses a class does not need to know all the details of the implementation of the class but need only know a much simpler description of how to use the class.
- Inheritance:
  - Process by which a new class (derived class) is created from another class called the base class (super class).
  - A derived class automatically has all the instance variable s and all the methods that the base class has, and can have additional methods and/or additional instance variables.
- Polymorphism
  - Technique to allow changes being made in the method definition for derived classes and apply those changes to the software written in the base class (super class).
  - Three types of polymorphism:
    i. Overloaded methods
    ii. Overridden methods
    iii. Dynamic (or late) binding methods

B. ARRAY

- Array used to store a collection of data - more useful to think of an array as a collection of variables of the same type.
- Declare single-dimensional array:

  Syntax: `elementType arrayName[]` OR

  `ElementType[] arrayName`

  Example:

```
char[] alphabet;
int[] list;
double[] price;
```

- Create single-dimensional array:

  Syntax: `arrayName = new elementType[SIZE]`

  Example:

```
alphabet = new char[SIZE];
list = new int[SIZE];
price = new double[SIZE];
```

- Declare and create single-dimensional array:

  Syntax: `elementType[] arrayName = new elementType[SIZE]`

  Example:

```
char[] alphabet = new char[SIZE];
int[] list = new int[SIZE];
double[] price = new double[SIZE];
```

Example:

Java Program Segment

```
final int SIZE = 5;
int[] array = new int[SIZE];

Scanner read = new Scanner(System.in);

System.out.println("Enter " + SIZE + " integer into array..");

for(int i=0; i<array.length; i++)
{
      System.out.print("array[" + i + "] <- ");
      array[i] = read.nextInt();
}

System.out.println();
System.out.println("Value entered into array");

for(int i=0; i<array.length; i++)
{
      System.out.println("array[" + i + "] -> " + array[i]);

      }
```

C. CLASSES AND OBJECT

- **Object**
  - An object represents an entity in the real world that can be distinctly identified.
  - An object has a unique identity, state and behaviour:
    - **State** – also known as its properties or attributes is represented by data fields with their current values. Example: Circle – radius, rectangle – width & height
    - **Behaviour** – also known as its action is defined by methods. To invoke a method on an object is to ask the object to perform an action. Example: circle – getArea().
  - Objects of the same type are defined using a common class. Object is an instance of a class.

- **Class**
  - Class is a template, blueprint or contract that defines what an objects data fields and methods will be.
  - A Java class uses variables to define data fields and methods to define actions.
  - Class provides methods of a special type, known as constructors, which are invoked to create new object.
  - A constructor can performs any action, but constructors are designed to perform initializing actions, such as initializing the data fields of objects.
  - There are 2 types of classes:
    i. ***Predefined*** classes, i.e. Scanner, String, Character, StringBuilder/StringBuffer,Math, Arrays, File, and etc.- classes that contained predefined method.
    ii. ***User-defined*** clasess – classes defined by programmer / user

Predefined Classes

- **Scanner Class**
  - Create object to read input from System.in class (*System.in is a class refer to the standard input device*).

Example:

Program

```
/*This program demonstrates the use of Scanner Predefined
* Class
*/

package pkgCSC305;

import java.util.Scanner;

public class ScannerClass {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.println("Enter any integer number : ");
        int number = in.nextInt();


        System.out.println("You have entered " + number);

    }
}
```

Makes the Scanner class available to program

Scanner class

Object

Method in Scanner

- `new Scanner(System.in)` – creates an object of the `Scanner` type.
- `Scanner in` – declares that `in` is a variable whose type is `Scanner`.
- `Scanner in = new Scanner(System.in)` creates a `Scanner` object and assigns its reference to the variable `in`.
- An object may invoke its methods. To invoke a method on object is to ask the object to perform a task.
- List of methods for Scanner objects:

Method	Description
nextByte()	Reads an integer if the byte type
nextShort()	Read an integer of the short type
nextInt()	Read an integer of the int type
nextLong()	Read an integer of the long type
nextFloat()	Read an integer of the float type
nextDouble()	Read an integer of the double type
next();	Read a string that ends before a whitespace character
nextLine();	Read a line of text (i.e. a string ending with the *Enter key* pressed). *To avoid input error, do not use after this method after all the other methods in Scanner class*

- **String Class**
  - Reference type for variables that represent string (sequence of characters)

Constructing a String

Example:

Program	Output

/*This program demonstrates the use of
* String Predefined Class
*/
package pkgCSC305;
 	
public class PredefinedClass {
 	
public PredefinedClass() {
}
 	
publicstaticvoid main(String[] args) {
 	
String msg = "Hello ";
String name = "Roslan ";
 	
System.out.println(msg + name);
 	
}
}
	Hello Roslan

String class → `String msg = "Hello ";`

`System.out` — Class; `println` — Method

String Comparison

Methods	Effect
s1.equals(s2)→boolean	Returns true if s1 equal to s2
s1.equalsIgnoreCase(s2)→boolean	Returns true if s1 equal to s2 case insensitive
s1.compareTo(s2)→ int	Returns 0 (s1 and s2 are indentical), < 0 (s1 less than s2), > 0 (s1 greater than s2)
s1.compareToIgnoreCase(s2)→ int	Returns 0 (s1 and s2 are indentical), < 0 (s1 less than s2), and > 0 (s1 greater than s2) case insensitive
s1.startWith(prefix)→ boolean	Returns true if s1 starts with the specified prefix
s1.endWith(suffix) → boolean	Returns true if s1 ends with the specified suffix

Example of string comparison:

i. s1.equals(s2 : String) : boolean and **s1.equalsIgnoreCase(s2 : String) : boolean**

Java Program Segment

```
String s1 = "Apple";
String s2 = "Apple";
String s3 = "aPPLe";

System.out.println("s1 : Apple");
System.out.println("s2 : Apple");
System.out.println("s3 : aPPLe");

if (s1.equals(s2))
      System.out.println("Case sensitive : " + s1.equals(s2) + " : s1 and
            s2 are identical");

if (s1.equals(s3))
      System.out.println("Case sensitive : " + s1.equals(s3) + " : s1 and
            s3 are identical");

else
      System.out.println("Case sensitive : " + s1.equals(s3) + " : s1 and
            s3 are not identical");

if (s1.equalsIgnoreCase(s3))
      System.out.println("Case insensitive : " + s1.equalsIgnoreCase(s3) +
            " : s1 and s3 are identical");
```

Output

```
s1 : Apple
s2 : Apple
s3 : aPPLe
Case sensitive : true : s1 and s2 are identical
Case sensitive : false : s1 and s3 are not identical
Case insensitive : true : s1 and s3 are identical
```

ii. s1.compareTo(s2 : String) : int and s1.equalsIgnoreCase(s2 : String) : int

Java Program Segment

```
String s1 = "Orange";
String s2 = "oRaNGe";

System.out.println("First Comparison : Case Sensitive");

if (s1.compareTo(s2) == 0)
      System.out.println(s1 + " equal to " + s2);
else if (s1.compareTo(s2) > 0)
      System.out.println(s1 + " greater than " + s2);
else
      System.out.println(s1 + " less than " + s2);

System.out.println();

System.out.println("Second Comparison -  Case insentive");

if (s1.compareToIgnoreCase(s2) == 0)
      System.out.println(s1 + " equal to " + s2);
else if (s1.compareToIgnoreCase(s2) > 0)
      System.out.println(s1 + " greater than " + s2);
else
      System.out.println(s1 + " less than " + s2);
```

Output

```
First Comparison : Case Sensitive
Orange less than oRaNGe

Second Comparison -  Case insentive
Orange equal to oRaNGe
```

iii. s1.equals(s2 : String) : boolean and s1.equalsIgnoreCase(s2 : String) : boolean

Java Program Segment

```
String pre = "App";
String suf = "nge";
String s1 = "Apple";
String s2 = "Orange";

if(s1.startsWith(pre))
      System.out.println(s1 + " starts with prefix " + pre);
else
      System.out.println(s1 + " does not starts with prefix " + pre);

if (s2.endsWith(suf))
      System.out.println(s2 + " ends with suffix " + suf);
else
      System.out.println(s2 + " does not ends with suffix " + suf);
```

Output

```
Apple starts with prefix App
Orange ends with suffix nge
```

String Length, Characters, and Combining Strings

Methods	Effect
s1.equals() → int	Returns the number
s1.charAt(index) → char	Returns the character at the specified index from this string
s1.concat(s2) → String	Returns a new string that concatenates this string with string s1

Example:

Java Program Segment

```
String s1 = "Apple";
String s2 = "Sweet ";

int lengthS1 = s1.length();;

System.out.println("The length of string " + s1 + " is " + lengthS1);

char c = s1.charAt(0);
System.out.println("Index 0 of " + s1 + " is character " + c);

String s3 = s2.concat(s1);
System.out.println(s2 + "+ " + s1 + " becomes " + s3);
```

Output

```
The length of string Apple is 5
Index 0 of Apple is character A
Sweet + Apple becomes Sweet Apple
```

String Length, Characters, and Combining Strings

Methods	Effect
s1.substring(beginIndex) → String	Returns the string's substring that begins with the character at the specified beginIndex and extends to the end of the string
s1.substring(beginIndex, endIndex) → String	Returns the string's substring that begins at the specified beginIndex and extends to the character at index endIndex – 1

Example:

Java Program Segment

```
String s1 = "Sweet Mango Cafe";
String s2 = s1.substring(6);
String s3 = s1.substring(0,11);

System.out.println("String s1.substring(6) of " + s1 + " is " + s2);
System.out.println("String s1.substring(0,11) of " + s1 + " is " + s3);
```

Output

```
String s1.substring(6) of Sweet Mango Cafe is Mango Cafe
String s1.substring(0,11) of Sweet Mango Cafe is Sweet Mango
```

Obtaining substrings

Example:

Java Program Segment

```
String s1 = "Sweet Mango Cafe";
String s2 = s1.substring(6);
String s3 = s1.substring(0,11);

System.out.println("String s1.substring(6) of " + s1 + " is " + s2);
System.out.println("String s1.substring(0,11) of " + s1 + " is " + s3);
```

Output

```
String s1.substring(6) of Sweet Mango Cafe is Mango Cafe
String s1.substring(0,11) of Sweet Mango Cafe is Sweet Mango
```

Converting, Replacing, and Splitting Strings

Methods	Effect
s1.toLowerCase() → String	Returns a new string with all characters converted to lowercase
s1.toUpperCase() → String	Returns a new string with all characters converted to uppercase
s1.trim() →String	Returns a new string with blank characters trimmed on both side
s1.replace(oldChar, newChar) → String	Returns a new string that replaces the all matching characters in this string with the new character
s1.replaceFirst(oldString, newString) → String	Returns a new string that replaces the first matching substring in this string with the new string
s1.replaceAll(oldString, newString) → String	Returns a new string that replaces all matching substring in this string with the new string
s1.split(delimiter) → String[]	Returns an array of strings consisting of the substrings split by the delimiter

Example:

Java Program Segment	Output

`String s = "Welcome";`
`String s2 = "  Welcome   ";`
`String s3 = "Wel#co#me";`
 	
`System.out.println(s.toLowerCase());`
`System.out.println(s.toUpperCase());`
`System.out.println(s2.trim());`
`System.out.println(s.replace('e', 'A'));`
`System.out.println(s.replaceFirst("e", "AB"));`
`System.out.println(s.replace("el", "AB"));`
 	
`String[] tokens = s3.split("#");`
`for (int i=0; i<tokens.length; i++)`
`System.out.print(tokens[i] + " ");`
	
	welcome
	WELCOME
	Welcome
	WAlcomA
	WABlcome
	WABcome
	
	Wel co me

Finding a Character or Substring in a String

Methods	Effect
s1.indexOf(ch) → int	Returns the index of the first occurrences of ch in the string, returns -1 if not matched
s1. indexOf (ch, fromIndex) → int	Returns the index of the first occurrence of ch after fromIndex in the string, returns -1 if not matched
s1. indexOf (str) → int	Returns the index of the first occurrence of string s in this string, returns -1 if not matched
s1.indexOf (str, fromIndex) → int	Returns the index of the first occurrence of string s in this string after fromIndex, returns -1 if not matched
s1.lastIndexOf(ch) →int	Returns the index of the last occurrence of ch in the string, returns -1 if not matched
s1.lastIndexOf(ch, fromIndex) → int	Returns the index of the last occurrence of ch before fromIndex in this string, returns -1 if not matched
s1. lastIndexOf(str) → int	Returns the index of the last occurence of string s, returns -1 if not matched
s1. lastIndexOf(str, fromIndex) → int	Returns the index of the last occurrence of string s before fromIndex, returns -1 if not matched

Example:

Java Program Segment	Return/Output
`String s1 = "Welcome to Java";`	
`System.out.println(s1.indexOf('W'));`	0
`System.out.println(s1.indexOf('e'));`	4
`System.out.println(s1.indexOf('o',5));`	9
`System.out.println(s1.indexOf("come"));`	3
`System.out.println(s1.indexOf("Java", 5));`	11
`System.out.println(s1.indexOf("java", 5));`	-1
`System.out.println(s1.lastIndexOf('W'));`	0
`System.out.println(s1.lastIndexOf('e'));`	9
`System.out.println(s1.lastIndexOf('o',5));`	4
`System.out.println(s1.lastIndexOf("come"));`	3
`System.out.println(s1.lastIndexOf("Java", 5));`	-1
`System.out.println(s1.lastIndexOf("java", 5));`	11

Conversion between Strings and Arrays

- String are not array but string can be converted into an array and vice versa

i. toCharArray

- to convert string to an array of characters

Example:

Java Program Segment	Output

`String text = "Java";`
`System.out.println(text);`
`char[] arrayChar = text.toCharArray();`
`for (int i=0; i <arrayChar.length; i++)`
`System.out.println("arrayChar[" + i + "]: " + arrayChar[i]);`
	`Java`
	`arrayChar[0]: J`
	`arrayChar[1]: a`
	`arrayChar[2]: v`
	`arrayChar[3]: a`

ii. s1.getChars(srcBegin: int, srcEnd: int, dst: char[], dstBegin: int)

- To copy a substring of the string from index **srcBegin** to **srcEnd-1** into a character array **dst**starting from index **dstBegin**.

Example:

Java Program Segment	Output

`String src = "CSC305";`
`char[] dst = {'I','T','C','3','0','5'};`
 	
`src.getChars(0,3,dst,0);`
`System.out.println(dst);`
	`CSC305`

iii. valueOf / String(char[])

- to convert an array of character to a String

Example:

Java Program Segment	Output

`String str = new String(new char[]{'J','a','v','a'});`
 	
`String str2 = String.valueOf(new char[]{'J','a','v','a'});`
 	
`System.out.println(str);`
`System.out.println(str2);`
	`Java`
	`Java`

iv. *Converting Characters and Numeric Values to String*

- The static valueOf method can be used to convert an array of characters into a string.

Methods	Effect
s1.valueOf(c: char) → String	Returns a string consisting of the character c
s1.valueOf(data: char[]) → String	Returns a string consisting of the characters in the array
s1.valueOf(d: double) → String	Returns a string representing the double value
s1.valueOf(f: float) → String	Returns a string representing the float value
s1.valueOf(i: int) → String	Returns a string representing the int value
s1.valueOf(l: long) → String	Returns a string representing the long value
s1.valueOf(b: boolean) → String	Return a string representing the boolean value

Problem: **Checking Palindromes**

- Reads the same forward and backwards

Java Program

```
package pkgCSC305;
import java.util.*;

public class Palindrome
{
	public static void main(String[] args) {

		Scanner read = new Scanner(System.in);
		System.out.print("Enter a string : ");
		String s = read.nextLine();

		if(isPalindrome(s))
		{
			System.out.println(s + " is a palindrome");
		}
		else
		{
			System.out.println(s + " is not a palindrome");
		}
	}
	public static boolean isPalindrome(String s)
	{
		int low = 0;
		int high = s.length() - 1;
		boolean  isTrue = true;

		while (low < high)
		{
			if (s.charAt(low) != s.charAt(high))
			{
				isTrue = false;
				break;
			}
			low++;
			high--;
		}
		return isTrue;
	}
}
```

- **Character Class**
  - Enable primitive data value of character to be treated as object
  - Example: Character character = new Character('a');

Methods / Constructor	Effect
c1.Character(value)	Construct a character object with char value
c1.charValue() → char	Returns the char value from this object
c1.compareTo(c2) → int	Compares this character with another
c1.equals(c2)→ boolean	Returns true if this character is equal another
Character.isDigit(c1) → boolean	Returns true if the specified character is digit
Character.isLetter(c1) → boolean	Returns true if the specified character is letter
Character.isLetterOrDigit(c1) → boolean	Returns true if the character is a letter or digit
Character.isLowerCase(c1) → boolean	Returns true if the character is a lowercase letter
Character.isUpperCase(c1) → boolean	Returns true if the character is an uppercase letter
Character.toLowerCase(c1) → char	Returns the lowercase of the specified character
Character.toUpperCase(c1) → char	Returns the uppercase of the specified character

Problem: Counting Each Letter in a String

Java Program

```
package pkgCSC305;
import java.util.Scanner;

public class CountingLetters {

	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);

		System.out.println("Enter a string : ");

		String s = input.nextLine();

		int [] counts = countLetter(s.toLowerCase());

		for (int i=0; i<counts.length; i++)
		{
			if (counts[i] != 0)
				System.out.println((char)('a' + i) + " appears " +
					counts[i] + ((counts[i] == 1) ? " time " : "
					times "));
		}
	}

	public static int[] countLetter(String s)
	{
		int [] counts = new int[26];

		for (int i=0; i<s.length(); i++)
		{
			if(Character.isLetter(s.charAt(i)))
				counts[s.charAt(i) - 'a']++;
		}

		return counts;
	}
}
```

- **StringBuilder/StringBuffer Class**
  - An alternative to the String class.
  - Can be used whenever String is used.
  - More flexible than String: add, insert, append new contents into a StringBuilder or a StringBuffer.
  - StringBuffer– methods for modifying buffer in it are synchronized.
  - StringBuffer – more efficient if it is accessed by single task.
  - Has three constructor andmore than 30 methods.

Example:

Java Program	Output

`StringBuilder strBuilder = new StringBuilder("Hello,");`
 	
`strBuilder.append(" ");`
`strBuilder.append("welcome");`
`strBuilder.append(" ");`
`strBuilder.append("to");`
`strBuilder.append(" ");`
`strBuilder.append("Java");`
 	
`System.out.println(strBuilder);`
 	
`strBuilder.insert(18, "HTML and ");`
 	
`System.out.println(strBuilder);`
 	
`strBuilder.delete(22,30);`
 	
`System.out.println(strBuilder);`
 	
`strBuilder.deleteCharAt(22);`
 	
`System.out.println(strBuilder);`
 	
`strBuilder.replace(18, 22, "Java");`
 	
`System.out.println(strBuilder);`
 	
`strBuilder.setCharAt(5, '!');`
 	
`System.out.println(strBuilder);`
 	
`strBuilder.reverse();`
 	
`System.out.println(strBuilder);`
	
	`Hello, welcome to Java`
	
	`Hello, welcome to HTML and Java`
	
	`Hello, welcome to HTMLa`
	
	`Hello, welcome to HTML`
	
	`Hello, welcome to Java`
	
	`Hello! welcome to Java`
	
	`avaJ ot emoclew !olleH`

Problem: **Ignoring Nonalphanumeric Characters When Checking Palindromes**

Java Program

```
package pkgCSC305;
import java.util.Scanner;

public class Palindrome2 {

	public static void main(String[] args)
	{
		Scanner read = new Scanner(System.in);

		System.out.println("Enter a string");
		String s = read.nextLine();

		System.out.println("Ignoring nonalphanumeric characters...");
		System.out.println("Is " + s + " a palindrome ? " +
			isPalindrome(s));
	}

	public static boolean isPalindrome(String s)
	{
		String s1 = filter(s);
		String s2 = reverse(s1);

	return s1.equals(s2);
	}

	public static String filter(String s)
	{
		StringBuilder strBuilder = new StringBuilder();

		for (int i=0; i<s.length(); i++)
		{
			if(Character.isLetter(s.charAt(i)))
				strBuilder.append(s.charAt(i));
		}

	return strBuilder.toString();
	}

	public static String reverse(String s)
	{
		StringBuilder strBuilder = new StringBuilder(s);
		strBuilder.reverse();

	return strBuilder.toString();
	}

}
```

- **Math Class**
  - Contains the methods needed to perform basic mathematical function
  - There are various class of methods under Math class, below are the few example:

i. Trigonometric Methods

Predefined methods	Effect / Value Returned
Math.sin(x)	Return the trigonometric sine of an angle in radians
Math.cos(x)	Return the trigonometric cosine of an angle in radians
Math.tan(x)	Return the trigonometric tangent of an angle in radians
Math.toRadians(x)	Convert the angle in degrees to an angle in radians
Math.toDegree(x)	Convert the angle in radians to an angle in degrees
Math.asin(x)	Return the angle in radians for the inverse of sin
Math.acos(x)	Return the angle in radians for the inverse of cos
Math.atan(x)	Return the angle in radians for the inverse of tangent

ii. Exponent Methods

Predefined methods	Effect / Value Returned
Math.exp(x)	Return e raise to the power of x (e^x)
Math.log(x)	Return the natural logarithm of x ($\ln(x) = \log_e(x)$)
Math.log10(x)	Return the base 10 logarithm of x ($\log_{10}(x)$)
Math.pow(x,y)	Return a raised to the power of b (a^b)
Math.sqrt(x)	Return the square root of x ($\sqrt{x}$) for x >= 0

iii. Rounding Methods

Predefined methods	Effect / Value Returned
Math.ceil(x)	x is rounded up to its nearest integer. This integer is returned as a double value
Math.floor(x)	x is rounded down to its nearest integer. This integer is returned as a double value
Math.rint(x)	x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double
Math.round(x)	x is rounded to its nearest integer

iv. min, max and abs methods

Predefined methods	Effect / Value Returned
Math.min(x,y)	Return the lower value of x and y
Math.max(x,y)	Return the higher value of x and y
Math.abs(x)	Return the absolute (positive) value of x

v. random method

Predefined methods	Effect / Value Returned
(int) (Math.random()*x)	Return a random number integer between 0 and x-*1*
x + (int) (Math.random()*y)	Return a random number integer between x and (x+y)-*1*

Example:

Java Program

```
package pkgCSC305;

public class RandomCharacter {

	/**Generate a random character between ch1,and ch2*/
	public static char getRandomCharacter(char ch1, char ch2) {
		return (char)(ch1 + Math.random() * (ch2 - ch1 + 1));
	}

	/**Generate a random lower case letter*/
	public static char getRandomLowerCaseLetter() {
		return getRandomCharacter('a', 'z');
	}

	/**Generate a random uppercase letter*/
	public static char getUpperCaseLetter() {
		return getRandomCharacter('A','B');
	}

	/**Generate a random digit character*/
	public static char getRandomDigitCharacter() {
		return getRandomCharacter('0','9');
	}

	/**Generate a random character*/
	public static char getRandomCharacter() {
		return getRandomCharacter('\u0000', '\uFFFF');
	}

	/**main() method*/
	public static void main(String[] args) {
		final int NUMBER_OF_CHARS = 175;
		final int CHARS_PER_LINE = 25;

		//print random characters between 'a' and 'z', 25 chars per
		//line
		for (int i=0; i<NUMBER_OF_CHARS; i++) {
			char ch = RandomCharacter.getRandomLowerCaseLetter();
			if ((i+1) % CHARS_PER_LINE == 0)
				System.out.println(ch);
			else
				System.out.print(ch);
		}
	}
}
```

- **Arrays Class**
  - **java.util.Arrays** class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements.
  - Static methods under Arrays class are:
    - java.util.Arrays.sort()
      - sort the whole array or partial array
    - java.util.Arrays.binarySearch()
      - Search for a key in an array – the array must be presorted in increasing order.
    - java.util.Arrays.equals()
      - to check whether two arrays are equal
      - two arrays are equal if they have the same contents.
    - java.util.Arrays.fill()
      - to fill in all or part of the array

i. java.util.Arrays.sort()

example:

Java Program	Output

`int[] numbers = {6,4,2,3,5};`
 	
`System.out.println("Before sort()");`
`for (int i=0; i<numbers.length; i++)`
`  System.out.print(numbers[i]+ " ");`
 	
`System.out.println();`
`java.util.Arrays.sort(numbers);`
`System.out.println("After sort()");`
`for (int i=0; i<numbers.length; i++)`
`  System.out.print(numbers[i] + " ");`
	`Before sort()`
	`6 4 2 3 5`
	`After sort()`
	`2 3 4 5 6`

ii. java.util.Arrays.binarySearch()

example:

Java Program	Output

`int[] numbers = {6,4,2,3,5};`
 	
`System.out.println("Before sort()");`
`for (int i=0; i<numbers.length; i++)`
`  System.out.print(numbers[i]+ " ");`
 	
`System.out.println();`
`java.util.Arrays.sort(numbers);`
`System.out.println("After sort()");`
 	
`for (int i=0; i<numbers.length; i++)`
`  System.out.print(numbers[i] + " ");`
	`Before sort()`
	`6 4 2 3 5`
	`After sort()`
	`2 3 4 5 6`

iii. java.util.Arrays.equals()

example:

Java Program	Output

`int[] list1 = {2,3,5,7,11};`
`int[] list2 = {2,3,5,7,11};`
 	
`System.out.println("Array List1: ");`
`for (int i=0; i<list1.length; i++)`
`System.out.print("  " + list1[i]);`
 	
`System.out.println();`
`System.out.println("Array List2: ");`
`for (int i=0; i<list2.length; i++)`
`System.out.print("  " + list2[i]);`
 	
`System.out.println();`
`System.out.println();`
 	
`if(java.util.Arrays.equals(list1, list2))`
`System.out.println("Array list1 and list2 are equal");`
	`Array List1:`
	`  2  3  5  7  11`
	`Array List2:`
	`  2  3  5  7  11`
	
	`Array list1 and list2 are equal`

iv. java.util.Arrays.fill()

example:

Java Program	Output

`int[] list1 = {1,2,3,4};`
 	
`java.util.Arrays.fill(list1, 3);`
 	
`for (int i=0; i<list1.length; i++)`
`  System.out.println(list1[i]);`
	`3`
	`3`
	`3`
	`3`

- **File Class**
  - A File object encapsulates the properties of a file or a path but does not contain the methods for creating a file or for reading/writing data from I/O file.
  - The File Class is intended to provide an abstraction that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion.
  - The File Class contains the methods for obtaining File properties and for renaming and deleting files.
  - In order to perform I/O you need to create objects using appropriate Java I/O classes. The objects contain the methods for reading/writing data from/to a file.
  - Read/write strings and numeric values from/to a text file can be performed by using Scanner and PrintWriter classes.

Predefined methods / Constructor	Effect / Value Returned
+.File(pathName: String)	Creates a File object for the specified path name. The path name may be a directory or a file.
+.File(parent: String, child: String)	Creates a File object for the child under the dirctory parent. The child may be a file name or a subdirectory
+.File(parent: File, Child: String)	Creates a File object for the child under the directory parent. The parent is a File object. In the preceeding constructor, the parent is a string.
+.exist(): boolean	Returns true if the file or the directory represented by the File object exists.
+.canRead(): boolean	Returns true if the file represented by the File object exists and can be written.
+.isDirectory(): boolean	Returns true if the File object represents a directory.
+.isFile(): boolean	Returns true if the File object represents a file.
+.isAbsolute: boolean	Returns true if the file represented in the File object is hidden. The exact definition of hidden is system dependent.
+.getAbsolutePath(): String	Returns the complete absolute file or directory name represented by the File object.
+.getCanonicalPath(): String	Returns the same as getAbsolutePath() except that it removes redundant names such as "." And ".." from the path name.
+.getName(): String	Returns the last name of the complete directory and file name represented by the File object. For example: new File("c:\\book\\test.dat").getName() returns test.dat.
+.getParent(): String	Returns the complete parent directory of the current directory or the file represented by the File object. For example: new File("c:\\book\\test.dat").getParent() returns c:\book.
+.lastModified(): long	Returns the time that the file was last modified.
+.length(): long	Return the size of the file, or 0 if it does not exist or if it is a directory.
+.listFile(): File[]	Returns the files under the directory for a directory File object.
+.delete(): boolean	Delete this file. The method returns true if the deletion succeeds.
+.renameTo (dest: File): boolean	Renames this file. The method returns true if the operation succeeds.

Writing data using PrintWriter

- java.io.PrintWrite class can be used to create a file and write data to a text file.
- Create a PrintWriter object for a text file as follows:

```
printWriter output = new printWriter(fileName);
```

Predefined methods / Constructor	Effect / Value Returned
+.PrintWriter(file:File)	Creates a PrintWriter object for the specified file object.
+.PrintWriter(fileName: String)	Creates a PrintWriter object for the specified file-name string.
+.print(s: String): void	Writes a string to the file.
+.print(c: char): void	Writes a character to the file.
+.print(cArray: char[]): void	Writes an array of characters to the file.
+.print(i: integer): void	Writes a integer to the file.
+.print(l: long): void	Writes a long to the file.
+.print(f: float): void	Writes a float to the file.
+.print(d: double): void	Writes a double to the file.
+.print(b: boolean): void	Writes a boolean to the file.
+.println() – prints a line separator	
+.printf() – prints using specified format	

Example:

Java Program

```
package pkgCSC305;

import java.io.FileNotFoundException;

public class WriteData {

      public static void main(String[] args) throws FileNotFoundException{
            java.io.File file =  new java.io.File("score.dat");

            if (file.exists())
            {
                  System.out.println("File already exists");
                  System.exit(0);
            }

            java.io.PrintWriter output = new java.io.PrintWriter(file);
            output.print("Mohammad Mirza Rafiqi");
            output.print(" Bin Rasidi");
            output.print(3.78);
            output.close();
      }
}
```

i. Reading data using Scanner

- java.util.Scanner used to read strings and primitive values from the console.
- A Scanner breaks it input into tokens delimited by whitespace characters.
- To read from the keyboard, you create a Scanner for System.in as follows:

```
Scanner input = new Scanner (System.in);
```

- To read from a file, you create a Scanner for a file as follows"

```
Scanner input = new Scanner (new File (fileName);
```

Predefined methods / Constructor	Effect / Value Returned
+.Scanner(source: File)	Creates a scanner that produces values scanned from the specified file.
+.Scanner(source: String)	Creates a scanner that produces values scanned from the specified string.
+.close()	Closes this scanner.
+.hasNext(): boolean	Returns true if this scanner has more data to be read.
+.next(): String	Returns next token as a string from this scanner.
+.nextLine(): String	Returns a line ending with the line separator from this scanner.
+.nextByte(): byte	Returns next token as a byte from this scanner.
+. nextShort(): short	Returns next token as a short from this scanner.
+.nextInt(): int	Returns next token as an integer from this scanner.
+.nextLong(): long	Returns next token as a long from this scanner.
+.nextFloat(): float	Returns next token as a float from this scanner.
+.nextDouble(): double	Returns next token as a double from this scanner.
+.useDelimiter(patter: String): Scanner	Set this scanner's delimiting pattern and returns this scanner.

Example:

Java Program

```
import java.util.Scanner;
import java.io.*;

public class ReadData {
	public static void main(String[] args) throws IOException {
		File file = new File("score.txt");
		Scanner input = new Scanner(file);

		while (input.hasNext()){
			String firstName = input.next();
			String secondName = input.next();
			String thirdName = input.next();
			String fourthName = input.next();
			String lastName = input.next();
			float score = input.nextFloat();

			System.out.println(firstName + " " + secondName + " " +
				thirdName + " " + fourthName + " " + lastName + " "
				+ score);
		}

		input.close();
	}
}
```

User-Defined Classes

- **Constructing Objects Using Constructors**
  - A constructor must have the same name as the class itself
  - Constructors do not have return type – not even void
  - Constructors are invoked using the new operator when an object is created. Constructors play the role of initializing objects.
  - A class normally provides a constructor without arguments (i.e. Circle()). Such constructor is referred to as a no-arg or no-argument constructor.
  - A class may be defined without constructors. Default constructor is provided automatically only if no constructors are explicitly defined in the class.

- **Accessing Objects via Reference Variables**
  - Newly created objects are allocated in the memory. The can be accessed via reference variables.
  - Reference variables are declared using the following syntax:

    className objectReferenceVariable;
  - Creates an object and assigns its reference to objectReferenceVariable:

    objectReferenceVariable = new className();
  - The following syntax is a declaration of an object reference variable, the creation of an object and the assigning of an object reference to the variable:

    className objectReferenceVariable = new className();

    Example:

    Circle c = new Circle();
  - The variable c holds a reference to Circle object.

- **Accessing an Object's Data and Methods**
  - After an object is created, its data can be accessed and its method invoked using the dot operator (.) also known as the object member access operator:
    - `objectReferenceVariable.dataField` – reference a data field in the object.
    - o`bjectReferenceVariable.method(arguments)` – invokes method on the object.
  - Example:
    - `c1.radius`
      - References the radius in c1
      - data field `radius` is referred to as an *instance variable*, because it is independent on a specific instance.

- `c1.getArea()`
  - Invokes the getArea() method on c1. Methods are invoked as operations on objects.
  - Method getArea() is referred to as instance method, because it can be invoke only on a specific instance.
  - The object on which an instance method is invoked is called a calling object.

Example 1: Defining Classess, Creating Objects And Accessing Object's Data and Methods

Java Program

```
package pkgCSC305;

class Circle{

	double radius; // field or attribute

	Circle(){ //constructor method with default radius
		radius = 1.0;
	}

	Circle(double newRadius){ //constructor with a specified radius
		radius = newRadius;
	}

	double getArea(){
		return radius * radius * Math.PI;
	}
}

public class TestCircle {

	public static void main(String [] args){

	Circle c1 = new Circle(); //create a circle with radius 1.0
	System.out.println("The area of the c1 of radius " + c1.radius + " is
		" + c1.getArea());

	Circle c2 = new Circle(25); //create a circle with radius 25
	System.out.println("The area of the c2 of radius " + c2.radius + " is
		" + c2.getArea());

	c2.radius = 100; // modify circle radius for object c2
	System.out.println("The area of the c2 of radius " + c2.radius + " is
		" + c2.getArea());

	c2 = new Circle(32); //create a circle with radius 25
	System.out.println("The area of the c2 of radius " + c2.radius + " is
		" + c2.getArea());
	}
}
```

- **Static Variables, Constants, and Methods**
  - Static variable:
    - In order for the instances of a class to share data, use static variables, also known as class variables.
    - Static variables store values for the variables in a common memory location.
    - Because of this common location, if one changes the value of a static variable, all objects of the same class are affected.
    - Declaration:

```
static int numberOfOjbect;
```

  - Static method:
    - Same as static variables.
    - Static methods can be called without creating an instance of the class.
    - Definition:

```
static int getNumberObject() { return numberOfObject; }
```

  - Declare constant:
    -

```
final static double PI = 3.1412;
```

Example:

Java Program

```
package pkgCSC305;

public class TestCircle2 {

	public static void main(String[] args) {
		System.out.println("Before creating objects");
		System.out.println("The number of Circle objects is " +
		Circle2.numberOfObjects);
		System.out.println("Default Area  : "
		+ new Circle().getArea());

		Circle2 c1 = new Circle2();

		System.out.println("\nAfter creating c1");
		System.out.println("c1: radius(" + c1.radius + ") and number of
		Circle objects " + c1.numberOfObjects);
		System.out.println("Area of c1 " + c1.getArea());

		Circle2 c2 = new Circle2(5);
		c1.radius = 9;

		System.out.println("\nAfter creating c2 and modifying c1");
		System.out.println("c1: radius(" + c1.radius + ") and number of
			Circle objects " + c1.numberOfObjects);
		System.out.println("Area of c1 " + c1.getArea());
		System.out.println("c2: radius(" + c2.radius + ") and number of
			Circle objects " + c2.numberOfObjects);
		System.out.println("Area of c2 " + c2.getArea());
	}
}

class Circle2{

	double radius;
	static int numberOfObjects = 0;

	Circle2()
	{
		radius = 1.0;
		numberOfObjects++;
	}

	Circle2(double newRadius)
	{
		radius = newRadius;
		numberOfObjects++;
	}

	static int getNumberObjects()
	{
		return numberOfObjects;
	}

	double getArea()
	{
		return radius * radius * Math.PI;
	}
}
```

- **Visibility Modifiers (visibility increases: private → none → protected → public)**
  - ***public***: `public` modifier means its visibility can be used for classes, methods and data fields to denote that they can be accessed from any other classes.
  - If there is no visibility modifier used, then by default the classes, methods and data fields are accessible by any class in the same package. This is known as package-private or package-access.
  - ***package***: `packages` can be used to organize classes. To do so, you need to add the following line as the first noncomment and nonblank statement in the program:

```
package packageName;
```

    Example:

```
package pkgCSC305;
```

  - ***private:*** `private` modifier makes method and data fields accessible only from within its own class.

```
package p1;

public class C1 {
public int x;
int y;
private int z;

public void m1() {}
void m2(){}
private void m3(){}
}
```

```
package p1;

public class C2 {
void aMethod() {
     C1 ob = new C1();
     Can access ob.x;
     Can access ob.y;
     Cannot access ob.z;

     Can invoke ob.m1();
     Can invoke ob.m2();
     Cannot invoke
            ob.m3();

   }
}
```

```
package p2;

public class C3 {
void aMethod() {
     C1 ob = new C1();
     Can access ob.x;
     Cannot access ob.y;
     Cannot access ob.z;

     Can invoke ob.m1();
     Cannot invoke ob.m2();
     Cannot invoke
      ob.m3();

   }
}
```

  - ***protected:*** to allow subclasses to access fields or methods defined in the superclass, but not allow non-sub classes to access the field and methods.

Modifier on Members in a class	Accessed from the same class	Accessed from the same package	Accessed from a subclass	Accessed from a different package
public	/	/	/	/
protected	/	/	/	-
default (none)	/	/	-	-
private	/	-	-	-

- **Data Field Encapsulation**
  - To prevent direct modification of data fields, you should declare the data fields **private**, using the private modifier. This is known as data field encapsulation.
  - A private data field cannot be accessed by an object from outside the class that defines the private field. But often a client needs to retrieve and modify a data field.
  - To make data field accessible, provide a *get* method to return its value. To enable a private data field to be updated, provide a set method to set a new value.

Example:

Java Program

```
package pkgCSC305;

public class TestCircle3 {

	public static void main(String[] args){

	Circle3 c1 = new Circle3(5.0);

	System.out.println("The area of the circle of radius " +
		c1.getRadius() + " is " + c1.getArea());

	c1.setRadius(c1.getRadius()* 1.1);

	System.out.println("The area of the circle of radius " +
		c1.getRadius() + " is " + c1.getArea());

	System.out.println("The number of object created is " +
		c1.getNumberOfObjects());
	}
}

class Circle3{

	private double radius = 1; //encapsulate radius;
	private static int numberOfObjects = 0; //encapsulate numberOfObjects

	public Circle3(){numberOfObjects++;}

	public Circle3(double newRadius){
		radius = newRadius;
		numberOfObjects++;
	}

	public double getRadius(){return radius;}

	public void setRadius(double newRadius){
		radius = (newRadius >= 0) ? newRadius : 0;
	}

	public static int getNumberOfObjects(){return numberOfObjects;}

	public double getArea(){
		return Math.pow(radius, 2) * Math.PI;
	}
}
```

- **Passing Objects to Methods**
  - like passing an array, passing an object is actually passing the reference of the object.

Example:

Java Program

```
package pkgCSC305;

class Circle4{
      private double radius = 1; //encapsulate radius;
      private static int numberOfObjects = 0; //encapsulate numberOfObjects

      public Circle4(){numberOfObjects++;}
      public Circle4(double newRadius){
            radius = newRadius;
            numberOfObjects++;
      }
      public double getRadius(){return radius;};
      public void setRadius(double newRadius){
            radius = (newRadius >= 0) ? newRadius : 0;
      }
      public static int getNumberOfObjects(){return numberOfObjects;}
      public double getArea(){return Math.pow(radius, 2) * Math.PI;}
}

public class PassObject {

      public static void printAreas(Circle4 c, int times){
            System.out.println("Radius \t\tArea");
            while (times >= 1)
            {
                  System.out.println(c.getRadius() + "\t\t" + c.getArea());
                  c.setRadius(c.getRadius()+1);
                  times--;
            }
      }

      public static void main (String [] args){
            Circle4 myCircle = new Circle4(1);

            int n = 5;
            printAreas(myCircle,n);

            System.out.println("\nRadius is " + myCircle.getRadius());
            System.out.println("n is " + n);    }


}
```

Output

```
Radius      Area
1.0         3.141592653589793
2.0         12.566370614359172
3.0         28.274333882308138
4.0         50.26548245743669
5.0         78.53981633974483

Radius is 6.0
n is 5
```

- **Array of Objects**
  - The following statement declares and creates array of 10 `obj` object.

```
ClassName[] obj = new ClassName[size];
```

    Example:

```
Circle[] circleArray = new Circle[10];
```

  - To initialize the`circleArray`, you can use a **`for`** loop like this one:

```
for (int i = 0; i <circleArray.length; i++) {
      circleArray[i] = new Circle();
}
```

Example:

Java Program

```
package pkgCSC305;

class Circle5{
      private double radius = 1; //encapsulate radius;
      private static int numberOfObjects = 0; //encapsulate numberOfObjects

      public Circle5(){numberOfObjects++;}
      public Circle5(double newRadius){
            radius = newRadius;
            numberOfObjects++;
      }
      public double getRadius(){return radius; }
      public void setRadius(double newRadius){
            radius = (newRadius >= 0) ? newRadius : 0;
      }
      public static int getNumberOfObjects(){return numberOfObjects;}
      public double getArea(){return Math.pow(radius, 2) * Math.PI;}
}

public class TotalArea {

      public static void main(String[] args){
            Circle5[] circleArray;
            circleArray = createCircleArray();
            printCircleArray(circleArray);
      }

      public static Circle5[] createCircleArray(){
            Circle5[] circleArray = new Circle5[5];

            for (int i = 0; i < circleArray.length; i++)
                  circleArray[i] = new Circle5(Math.random() * 100);

            return circleArray;
      }
```

```
	public static void printCircleArray(Circle5[] circleArray){
		System.out.printf("%-30s%-15s\n", "Radius", "Area");
		for (int i = 0; i < circleArray.length; i++) {
			System.out.printf("%-30s%-15s\n",
			circleArray[i].getRadius(), circleArray[i].getArea());
		}

		System.out.println("----------------------------------------
			-------------");
		System.out.printf("%-30s%-15s\n", "The total area of circle is
			", sum(circleArray));
	}
	public static double sum(Circle5[] circleArray){
		double sum = 0;
		for (int i = 0; i < circleArray.length; i++)
			sum += circleArray[i].getArea();
		return sum;
	}

}
```

Output

```
Radius                         Area
40.43061764438477              5135.356814405086
94.62579335893402              28129.948699229713
89.72514484194619              25291.710896656983
97.32597575533629              29758.25121328157
92.30989550748046              26769.877966284428
------------------------------------------------------------
The total area of circle is    115085.14558985777
```

D. THINKING IN OBJECT

- **Immutable Objects and Classes**
  - Content of object which cannot be changed once the object is created.
  - If the class is a mutable, then all its data filed must be private and it cannot contain public **set** methods for any data fields.
  - For a class to be immutable, it must meet the following requirements:
    - all data fields private;
    - no mutator methods;
    - no accessor method that returns a reference to a data field that is mutable.

Example:

Java Program

```
public class Student{

      private int id;
      private String name;

      public Student (int ssn, String newName){//Normal constructor
            id = ssn;
            name = newName;
      }
      public int getID(){ return id; }
      public String getName() { return name; }
}
```

- **Acessor and Mutator Methods**
  - **Accessor method**: allow you to obtain the data from a class object.
  - **Mutator method**: allow you to change the data in a class object.

Example:

Java Program

```
public class Student{
      private int id;
      private String name;
      public student(){ // default constructor
            id = 0;
            name = " ";
      }
      public Student (int ssn, String newName){// Normal constructor
            id = ssn;
            name = newName;
      }
      public int getID(){ return id; }    // Accessor method
      public String getName() { return name; }// Accessor method
      public void setName(String newName) { name = newName;}// mutator
                                                            // method
      public void setId (int newId) { id = newId; }// mutator method

}
```

- **The this Reference**
  - The `this` keyword is the name of a reference that refers to a calling object itself.
  - One of its common uses is to reference a class's hidden data fields.
  - A hidden static variable can be accessed simply by using the ClassName.StaticVariable reference. A hidden instance variable can be accessed by using the keyword this.

Example:

Java Program

```
package pkgCSC305;

public class TheThis {

public static void main(String [] args){

Foo f1 = new Foo(1,1);
System.out.println("i = " + f1.getI());
System.out.println("this.j = " + f1.getJ());

Foo f2 = new Foo();
f2.setI(2);
f2.setJ(2);

System.out.println("i = " + f2.getI());
System.out.println("this.j = " + f2.getJ());

Foo f3 = new Foo();
System.out.println("i = " + f3.getI());
System.out.println("this.j = " + f3.getJ());
System.out.println(Foo.i +  " " );
System.out.println(f3.getJ2() +  " " );

    }
}
class Foo {
      static int i = 0;
      int j = 0;

      public Foo(){}
      public Foo(int a, int b){
            i = a;
            j = b;
      }
      public int getI(){return i;}
      public int getJ(){return this.j;}
      public int getJ2(){return j; }
      void setI(int a) {i = a;}
      void setJ(int b) {this.j = b;}
}
```

Output

```
i = 1
this.j = 1
i = 2
this.j = 2
i = 2
this.j = 0
2
0
```

E. INHERITANCE AND POLYMORPHISM

- **Inheritance:** Is an important and powerful feature in Java for reusing software.
  - ***Superclasses***: also referred to as a parent class or a base class.
  - ***Subclass***: as extended class or derived class from superclass. It inherits accessible data fields and methods from its superclass and may also add new data fields and methods.
    - Subclass is not a subset of its superclass. In fact, a subclass usually contains more information and methods than its superclass.
    - Privated data fields in a superclass are not accessible outside the class; therefore they cannot be used directly in a subclass. They can, however, be accessed/mutated through public accessor/mutator if defined in the superclass.
    - Not all *is-a* relationship should be modeled using inheritance.
    - Inheritance is used to model the *is-a* relationship. Do not blindly extend a class just for the sake of reusing methods. A subclass and its superclass must have the *is-a* relationship.

- **Using the super Keyword**
  - The keyword super refers to the superclass of the class in which super appears.
  - It can be used in two ways:
    - To call a superclass constructor
    - To call a superclass method
  - The syntax to call a superclass constructor is : `super();` or `super(parameter);`

- The following points regarding inheritance were worthwhile to note:

  - A subclass is not a subset of its superclass. Usually, a subclass contains more information and methods that its superclass.
  - Private data fields in a superclass are not accessible outside the class. Therefore they cannot be used directly in a subclass. They can be accessed/mutated through public accessor/mutator if defined in the superclass.
  - Not all is-a relationship should be modeled using inheritance.
  - Inheritance is used to model the is-a relationship. A subclass and its superclass must have the is-a relationship.
  - Java does not allow multiple inheritance (C++ allow multiple inheritance).

Example:

Java Program (Base Class)

```
package pkgCSC305;

public class GeometricObject1 {
	private String color = "White";
	private boolean filled;
	private java.util.Date dateCreated;

	//default constructor
	public GeometricObject1(){ dateCreated =  new java.util.Date();}

	// normal constructor
	public GeometricObject1(String color, boolean filled){
		dateCreated = new java.util.Date();
		this.color = color;
		this.filled = filled;
	}

	public String getColor(){ //accessor
		return color;
	}


	public void setColor(String color) { //mutator
		this.color = color;
	}

	public boolean isFilled(){ //accessor
		return filled;
	}

	Public void setFilled(boolean filled){    //mutator
		this.filled = filled;
	}

	public java.util.Date getDateCreated(){
		return dateCreated;
	}

	public String toString(){
		return "created on " + dateCreated + "\ncolor: " + color + "
		and filled : " + filled;
	}
}
```

Java Program (Sub Class)

```
package pkgCSC305;

public class Circle7 extends GeometricObject1{
	private double radius;

	public Circle7(){}

	public Circle7(double radius){
		this.radius = radius;
	}
```

```
public Circle7(double radius, String color, boolean filled){
		this.radius = radius;
		setColor(color);
		setFilled(filled);
	}

	public double getRadius(){
		return radius;
	}

	public void setRadius(double radius){
		this.radius = radius;
	}

	public double getArea(){
		return Math.pow(radius, 2) * Math.PI;
	}

	public double getDiameter(){
		return 2 * radius;
	}

	public double getPerimeter(){
		return 2 * radius * Math.PI;
	}

	public void printCircle(){
		System.out.println("The circle is created " + getDateCreated()
		+ " and the radius is " + radius);
	}
}
```

Java Program (Main Class)

```
package pkgCSC305;

public class TestCircleRectangle {

	public static void main(String[] args) {

		Circle7 circle = new Circle7(1);
		System.out.println("A circle " + circle.toString());
		System.out.println("A circle " + circle.getRadius());
		System.out.println("A circle " + circle.getArea());
		System.out.println("A circle " + circle.getDiameter());
	}
}
```

Output

```
A circle created on Sat Jul 21 18:24:52 SGT 2012
color: White and filled : false
A circle 1.0
A circle 3.141592653589793
A circle 2.0
```

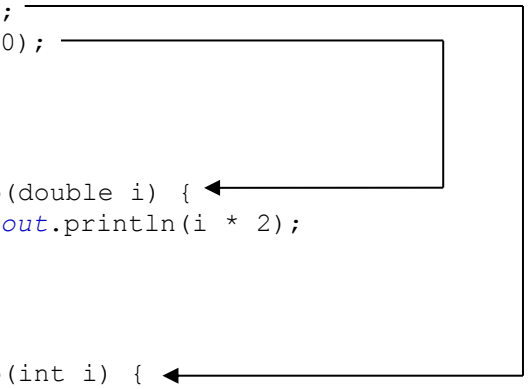
- **Polymorphism:** Is the capability of an action or method to do different things based on the object that is acting upon. This is the third basic principle of OOP. There are **THREE** types of polymorphism:

  i. **Overloaded Methods**: Are methods with the same name signature but either a different number of parameters or different types in the parameter.

Example:

Java Program

```
public class Test {
	public static void main(String[] args){
		A a = new A();
		a.p(10);
		a.p(10.0);
	}
}

class B {
	public void p(double i) {
		System.out.println(i * 2);
	}
}

class A extends B {
	public void p(int i) {
		System.out.println(i);
	}
}
```



ii. **Overriden Methods**: Are methods that are defined within an inherited or subclass. They have the same signature and the subclass definition is used.

Example:

Java Program

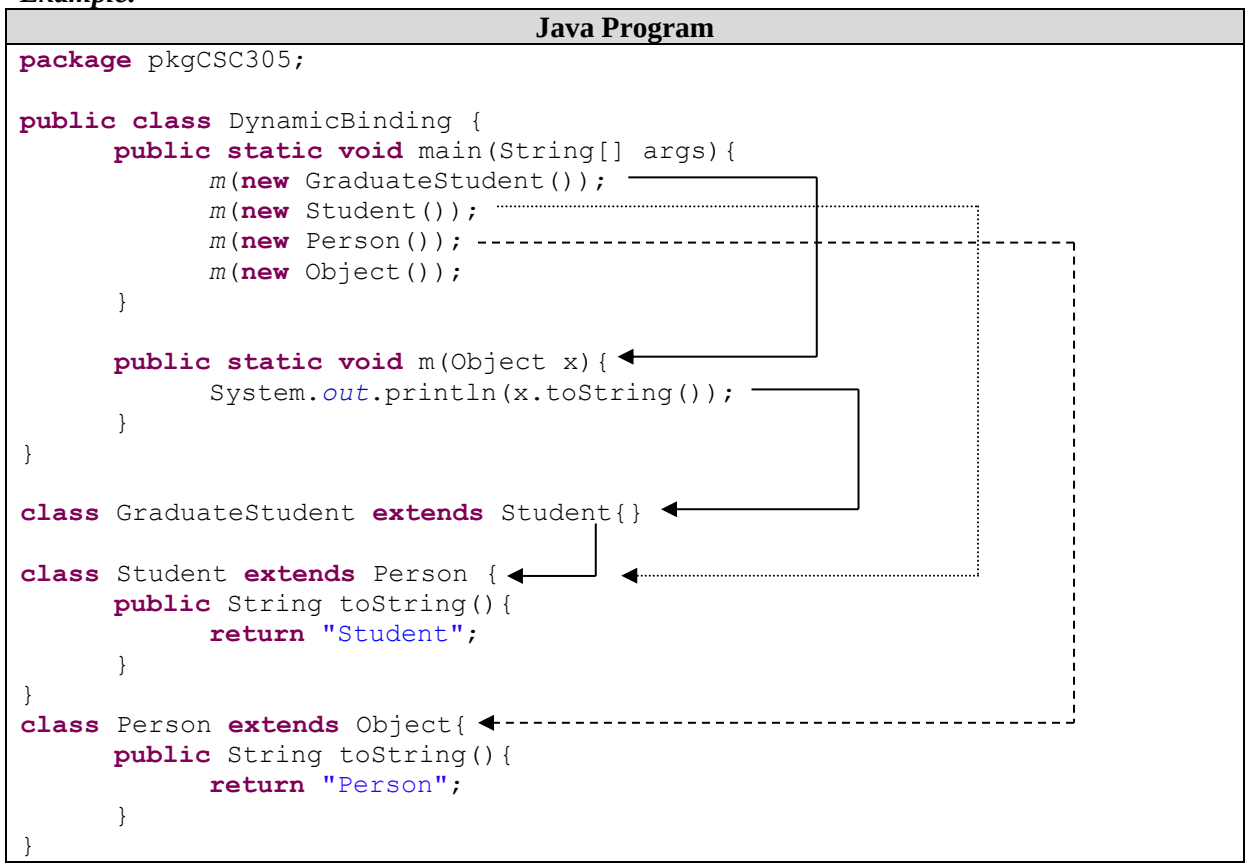
```
public class Test {
	public static void main(String[] args){
		A a = new A();
		a.p(10);
		a.p(10.0);
	}
}

class B {
	public void p(double i) {
		System.out.println(i * 2);
	}
}

class A extends B {
	public void p(double i) {
		System.out.println(i);
	}
}
```

iii. **Dynamic method binding**:Is the ability of a program to resolve references to subclass methods at runtime.

Example:

Java Program

```
package pkgCSC305;

public class DynamicBinding {
      public static void main(String[] args){
            m(new GraduateStudent());
            m(new Student());
            m(new Person());
            m(new Object());
      }

      public static void m(Object x){
            System.out.println(x.toString());
      }
}

class GraduateStudent extends Student{}

class Student extends Person {
      public String toString(){
            return "Student";
      }
}
class Person extends Object{
      public String toString(){
            return "Person";
      }
}
```



F. OBJECT COMPOSITION

- An object can contain another object. The relationship between the two is called composition.
- Composition is actually a special case of the aggregation relationship. Aggregation model has-a relationship and represents an ownership between two objects.
- The owner is called an aggregating object and its class an aggregating class.
- The subject object is called an aggregated object and its class an aggregated class.

Aggregated Class	Aggregating Class	Aggregated Class
`public class Name { .... }`	`public class Student { private Name name; private Address add; .... }`	`public class Address { .... }`

Example:

Java Program

```
package pkgCSC305;

class Engine {
      public String start() {return "Engine Start";}
      public String rev() {return "Engine Reverse";}
      public String stop() {return "Engine Stop";}
}

class Wheel {
      public int inflate(int psi) {return psi;}
}

class Window {
      public String rollup() {return "Window: Rool up";}
      public String rolldown() {return "Window: Roll Down";}
}

class Door {
      public Window window = new Window();
      public String open() {return "Door: Open";}
      public String close() {return "Door: Close";}
}

Public class Car {
      public Engine engine = new Engine();
      public Wheel[] wheel = new Wheel[4];
      public Door
      left = new Door(),
      right = new Door(); // 2-door
      public Car() {
            for(int i = 0; i < 4; i++)
                  wheel[i] = new Wheel();
      }
      public static void main(String[] args) {
            Car car = new Car();
            System.out.println(car.left.window.rollup());
            System.out.println("PSI for Wheel[0]: " +
                  car.wheel[0].inflate(72));
      }
}
```

G. ABSTRACT CLASSES AND INTERFACES

- ***Abstract class***: superclass that cannot have any specific instances.
- ***Abstract method***: method in abstract class that have no implementation. The implementation detail is in subclass.

Example:

Java Program (Abstract Class)

```
package pkgCSC305;

public abstract class GeometricObject {

      private String color = "White";
      private boolean filled;
      private java.util.Date dateCreated;

      public GeometricObject(){ dateCreated =  new java.util.Date();}
      public GeometricObject(String color, boolean filled){
      dateCreated = new java.util.Date();
            this.color = color;
            this.filled = filled;
      }

      public String getColor(){
            return color;
      }

      publicvoid setColor(String color) {
            this.color = color;
      }

      publicboolean isFilled(){
            return filled;
      }

      publicvoid setFilled(boolean filled){
            this.filled = filled;
      }

      public java.util.Date getDateCreated(){
            return dateCreated;
      }

      public String toString(){
            return "created on " + dateCreated + "\ncolor: " + color + "
                  and filled : " + filled;
      }

      Public abstract double getArea();   //abstract method
      public abstract double getPerimeter();
}
```

Java Program (Sub Class)

```
package pkgCSC305;

public class Circle8 extends GeometricObject{
	private double radius;

	public Circle8(){}

	public Circle8(double radius){
		this.radius = radius;
	}

	public Circle8(double radius, String color, boolean filled){
		this.radius = radius;
		setColor(color);
		setFilled(filled);
	}

	public double getRadius(){
		return radius;
	}

	public void setRadius(double radius){
		this.radius = radius;
	}

	public double getArea(){
		return Math.pow(radius, 2) * Math.PI;
	}

	publicdouble getDiameter(){
		return 2 * radius;
	}

	public double getPerimeter(){
		return 2 * radius * Math.PI;
	}

	public void printCircle(){
		System.out.println("The circle is created " + getDateCreated()
		+ " and the radius is " + radius);
	}
}
```

Java Program (Sub Class)

```
package pkgCSC305;

public class TestGeometricObject {

	public static void main(String[] args){
		GeometricObject geoObject = new Circle8(5);

		displayGeometricObject(geoObject);
	}
	public static void displayGeometricObject(GeometricObject object){
		System.out.println("The area is " + object.getArea());
		System.out.println("The perimeter is " +
			object.getPerimeter());
	}
}
```

- ***Interface***: is a classlike construct that contains only constants and abstract methods. It is similar to abstract class, but its intent to specify common behaviour for objects.
- ***Example of interface***:

```
public interface Edible {
      public abstract String howToEat();
}
```

Example:

Java Program (Sub Class)

```
package pkgCSC305;

class Animal {}

class Chicken extends Animal implements Edible{
      public String howToEat() {
            return "Chicken: Fry it";
      }
}

class Tiger extends Animal {}

abstract class Fruit implements Edible {}

class Apple extends Fruit {
      public String howToEat() {
            return "Apple: Make apple cider";
      }
}

class Orange extends Fruit {
      public String howToEat() {
            return "Orange: Make orange juice";
      }
}

interface Edible{
      public abstract String howToEat();
}

public class TestEdible {

      public static void main(String[] args) {
            Object[] objects = {new Apple(), new Orange(), new Chicken(),
                  new Tiger()};
            for (int i = 0; i < objects.length; i++)
                        if (objects[i] instanceof Edible)
                                    System.out.println(((Edible)
                                          objects[i]).howToEat());
      }
}
```

Output

```
Apple: Make apple cider
Orange: Make orange juice
Chicken: Fry it
```

REFERENCES:

Sebesta, W. R., (2010). ***Concepts of Programming Languages, Ninth Edition***. Pearson

Liang, D. Y., (2011).***Introduction to Java Programming***. Pearson

EXAMPLES OF PAST EXAM QUESTION

[Mar2012-B3]

```
public abstract class Article{

	private String title;
	protected double price;
	protected String publisher;
	protected String writer_name;

	public Article(String t, double pr, String p) {
	title = t;
	publisher = p;
	price = pr;
	}

	public String getTitle() {return title;}
	public double getPrice() {return price;}
	public String getName() {return writer_name;}

	public String toString() {
		return "Article title : " + title + "\nPrice : " + price +
		 "\nWrite Name : " + writer_name;
	}

	public abstract int getLength();
	public abstract double calcPrice();
}
```

a. Derive two subclasses named magazine and CD. These subclasses should have getLength() and calcPrice() methods.

Magazine: adds a page count (number of pages) and cost per page.

CD: adds a playing time (in minutes) and cost per minute.

getLength will return the number of pages or playing times (in minutes).

calcPrice will calculate the cost of book (number of pages * cost per pages) and the cost for CD is number of minutes * cost per minute and set its price in the Article class.

getCost will return the price.

Both subclasses have their normal constructors and a toString method each.

Answer:

```
import java.text.DecimalFormat;
public class CD extends Article {
	private int time;
	private double cost;

	public CD(String t, double p, String au, int m, double c)
	{
		super(t,p, au);
		time = m;
		cost = c;
	}

	public int getLength(){
		return time;
	}

	public double calcPrice(){
		price = cost * time;
		return price;
	}

	public String toString() {
		java.text.DecimalFormat dc = new DecimalFormat("0.00");
		return super.toString() + "\nPlaying time (minutes) : " + time
		 + "\n Cost per minute: RM" + dc.format(cost);
	}

}

public class Magazine extends Article {
	private int page;
	private double cost;

	public Magazine(String t, double p, String au, int m, double c)
	{
		super(t,p, au);
		page = m;
		cost = c;
	}

	public int getLength(){
		return page ;
	}

	public double calcPrice(){
		price = cost * page;
		return price;
	}

	public String toString() {
		java.text.DecimalFormat dc = new DecimalFormat("0.00");
		return super.toString() + "\nPlaying time (minutes) : " + page
	+ "\n Cost per minute: RM" + dc.format(cost);
	}

}
```

b. Write a complete application class called ArticleApp that performs each of the following in sequence:

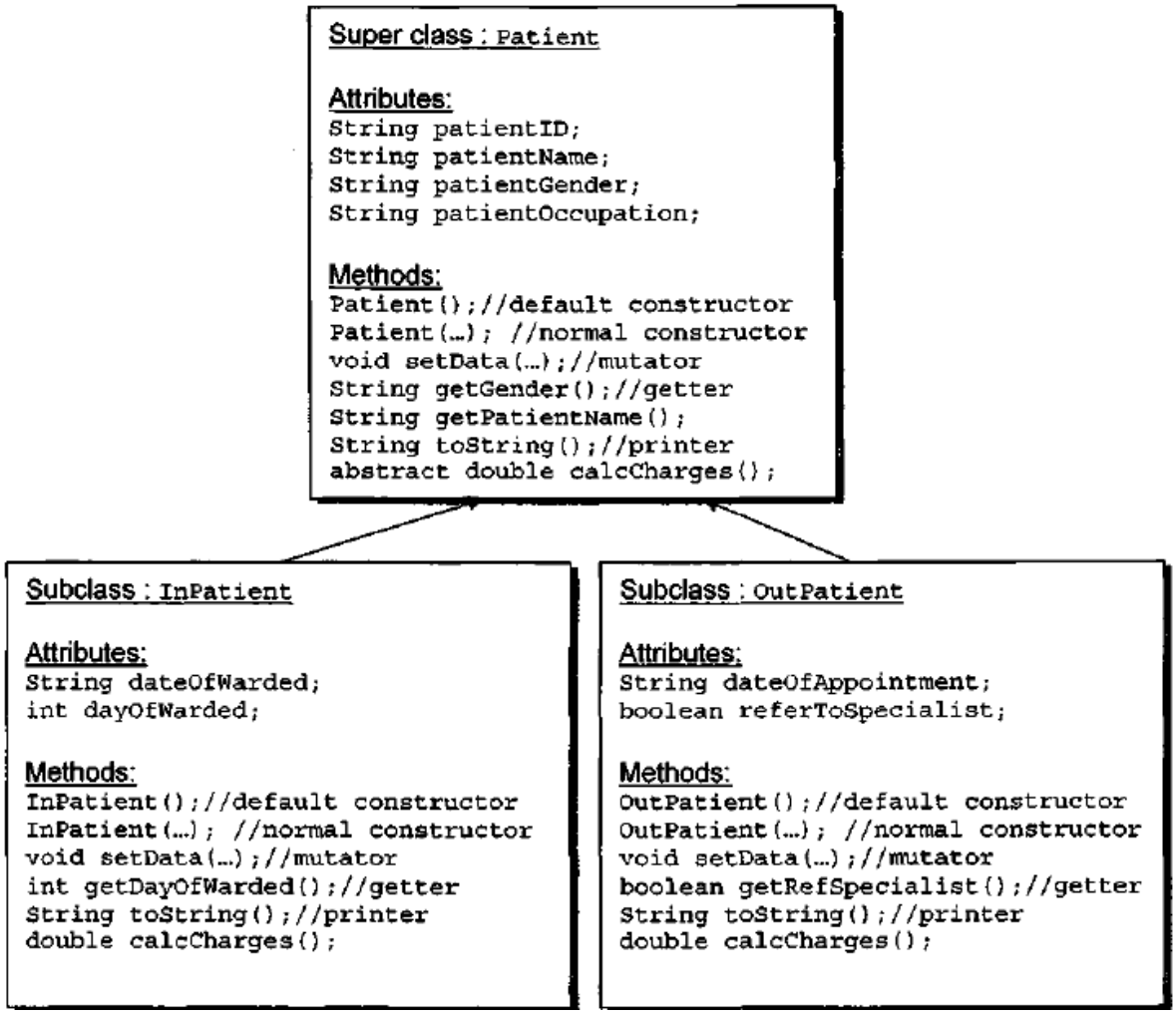
- Read the total number of published article.
- Declare an array of object to hold the published materials using polymorphism.
- For each published material, read in the type of publication (Magazine or CD) and the properties.

Answer:

```
import java.util.*;
import javax.swing.*;


public class ArticleApp {

	public static void main(String[] args) {
		int b = 0;
		double total = 0;

		int ask = Integer.parseInt(JOptionPane.showInputDialog("Enter number
				of Artcile :"));
		Article AA[] = new Article[ask];

		for (int i=0; i<AA.length;i++)
		{
			String a = JOptionPane.showInputDialog("Enter the Title");
			String type = JOptionPane.showInputDialog("Enter type of
				Article");
			char ty = type.charAt(0);
			String c = JOptionPane.showInputDialog("Enter the Write Name");


			if (ty == 'm' || ty == 'M')
			{
				String j = JOptionPane.showInputDialog("Enter the
					number of pages : ");
				int w = Integer.parseInt(j);
				String h = JOptionPane.showInputDialog("Enter the cost
						per pages: RM");
				double m = Double.parseDouble(h);
				double f = 0;
				AA[i] = new Magazine(a,f,c,w,m);
			}

			if (ty == 'c' || ty == 'C')
			{
				String j = JOptionPane.showInputDialog("Enter the playing
					time (minutes) : ");
				int w = Integer.parseInt(j);
				String h = JOptionPane.showInputDialog("Enter the cost
					per minutes: RM");
				double m = Double.parseDouble(h);
				double f = 0;
				AA[i] = new Magazine(a,f,c,w,m);
			}
			AA[i].calcPrice();
		}
	}
}
```

[Sept2011-B3]

Given the following diagram are the class of Patient as super class and InPatient and OutPatient as subclasses.



Answer the following questions based on the above diagram.

a) Write the normal constructor for both classes.

Answer:

```
InPatient(String patientID, String patientName, String patientGender, String
       patientOccupation, String dateOfWarded, int dayOfWarded)
{
       super(patientID, patientName, patientGender, patientOccupation);
       this.dateOfWarded = dateOfWarded;
       this.dayOfWarded = dayOfWarded;
}

OutPatient String patientID, String patientName, String patientGender, String
       patientOccupation, String dateOfAppointment, boolean referToSpecialist)
{
       super(patientID, patientName, patientGender, patientOccupation);
       this.dateOfAppointment;
       this. referToSpecialist;
}
```

b) Write the definition for abstract method from both subclasses. Given for out patient the charges will be incurred when the patient will be referred to specialist doctor. The charges are RM20. For in patient, charges depend on the number of days warded. The charges are RM 10 per day.

Answer:

```
double calcCharges()
{
       return getDayOfWarder * 10;
}

double calcCharges()
{
       double charge = 0;
       if (getRefSpecialist() ==  true)
              charges =  20;
       return charge;
}
```

c) Write a program fragment to determine total charges that have been collected from all patients.

Answer:

```
double totalCharge = 0;
for (int i=0; i<patient.length; i++)
       totalCharge = totalCharge + patient[i].calcCharge();
System.out.println("\nTotal charges that have been collected, RM") <<
totalCharge.
```

d) Count number of patient who have been warded more than 5 days.

Answer:

```
int count=0;
for (int j=0; j<patient.length; j++)
{
       if(patient[j] instanceOf InPatient) {
              InPatient temp = (InPatient) patient[j];
              if (temp.getDayOfWarded() > 5)
                     count++;
       }
}

System.out.print("\nNumber of patients warded for more than 5 days " + count);
```

e) Lis all males's name that has been referred to the specialist doctor.

Answer:

```
for (int i=0; i<patient.length; i++)
{
       if (patient[j] instanceOf OutPatient) {
              OutPatient temp = (OutPatient) patient[j];
              if ((temp.getRefSpecialist() == true) &&
                     (temp.getGender().equals("male")))
              System.out.print("\nName : " +temp.getPatientName());
       }
}
```

TUTORIAL