Assignment 7

① Explain Encapsulation with an Example.

→ Encapsulation in Java is a process of wrapping code & data together into a single unit. for Example, a capsule which is mixed of several medicines.

→ "Java Bean" class is the example of a fully encapsulated class. & Encapsulation is also called as data hiding / data binding.

Ex: Public class Student {

Private String name;

Public String getName() { // getter method

return name; }

Public void setName (String name) {

this.name = name; }

}

Public class StudentTest {

Public static void main (String[] args) {

Student S1 = new student();

S1.setName ("Archana");

S1.getName();

}

O/p: Archana.

② Explain naming convention for Java Bean for getter setter methods for boolean & non-boolean types.

→ In Java, getter & setter are two conventional methods that are used for retrieving & updating value of a variable.

Variable declaration	Getter method	Setter method
① int quantity	int getQuantity()	void setQuantity(int qty)
② String firstName	String getFirstName()	void setFirstName(String fname)
③ Date birthday	Date getBirthday()	void setBirthday(int bornDate).
④ boolean rich	boolean isRich() boolean getRich()	void setRich(boolean rich)

Eg: non-boolean type

→ Ex: Private String name;

Public void setName(String name)
{
}

Public String getName() { }

Eg: boolean type

If the variable is of the type boolean, then the getter's name can be either isXxx() or getXxx(), but the former naming is preferred.

Ex: Private boolean single;

Public String isSingle() { }

Public void setSingle(boolean single)
{ }

③ difference b/w Comparable & Comparator Interfaces

Comparable	Comparator
① Comparable provides a single sorting sequence. mean, we can sort the collection on the basis of a single element such as id, name, & price.	① Comparator provides multiple sorting sequence. mean. we can sort the collection on the basis of multiple elements such as id, name & price etc.
② It affects the original class, i.e. the actual class is modified	② doesn't affect the original class. i.e, the actual class is not modified.
③ Provides CompareTo() method to sort element	③ provides Compare() method to sort elements
④ is present in java.lang Package	④ is present in the Java. util Package.
⑤ We can sort the list elements of Comparable type by Collections. sort (List) method.	⑤ We can sort the list elements of Comparator type. by Collections. sort (List, Comparator) method.

④ Explain difference between default Constructor & parameterized constr/.

default Constructor	Parameterized Constructor
1) A constructor that is automatically generated by the compiler in the absence of any programmer defined constructor	1) A constr/ that is created by the programmer with one or more parameters to initialize the instance variables of a class.

② ~~If the programmer has not~~ The access modifier of default constr/ is always the same as a class modifier but this rule is applicable only for "Public" & "default" modifiers.

③ To assign default values to the newly created objects is the main responsibility of default constr/

② A para/ constr/ is written explicitly by a progr-ammer.

③ The purpose of a parameter/ constr/ is to assign user-wanted specified values to the instance variables of diff/ objects

⑤ Explain importance of equals & hashcode methods.

→ Java equals() & hashcode() methods are present in object class. So every Java class gets the default implementation of equals() & hashcode()

implementation :

① if o1.equals(o2), then o1.hashcode() == o2.hashcode() should always be true.

② if o1.hashcode() == o2.hashcode is true, it doesn't mean that o1.equals(o2) will be true.

equals() : ① for any object x, x.equals(x) should return true.

② for any two object x & y, x.equals(y) should return true if & only if y.equals(x) returns true

③ for multiple objects x, y & z, if x.equals(y) return true & y.equals(z) return true, then x.equals(z) should return true

④ Object class equals() method implementation return true only when both the reference are pointing to same object.

hashcode() ⑤ Multiple invocations of x.equals(y) should return same result, unless any of the object properties is modified that is being used in the equals() method implementation.

hashcode(): is a native method & return the integer hashcode value of the object. The general contract of hashcode() method is

→ Multiple invocations of hashcode() should return the same integer value, unless the object property is modified that is being used in the equals() method.

→ An object hashcode value can change in multiple executions of the same application.

→ If two objects are equal according to equals() method, then their hash code must be same.

→ If two objects are unequal according to equals() method, their hash code are not required to be different. Their hash code value may or may-not be equal.

⑥ why == should not be used to compare objects? How else should we check for equality.

→ "==" compares object references with each other &, not their literal values. If both the variables point to same object, it will return true, so

Ex:- String s1 = new String ("Hello");
String s2 = new String ("Hello");

Here s1 == s2 will return false as both are different objects

when you use equals(), it will compare the literal values of the content & give the results

(7) Explain what is the output:

String s = "Abc";
String s1 = new String ("abc");
S.O.P (s == s1)
S.O.P (s.equals (s1));
S.O.P (s.equalsIgnoreCase (s1));

Explain why you get true for only the last case

→ equalsIgnoreCase() in Java compares two String irrespective of the case (lower & upper) of the string.

(8) This method return true if the arguments is not null & it represents an equivalent string ignoring case, else false