23/30

Mid-term Answersheet

Name: Syeda Nowshin Ibnat

ID: 17183103020

Intake: 39 section: 01

course title: Advanced programming

course code: CSE 331

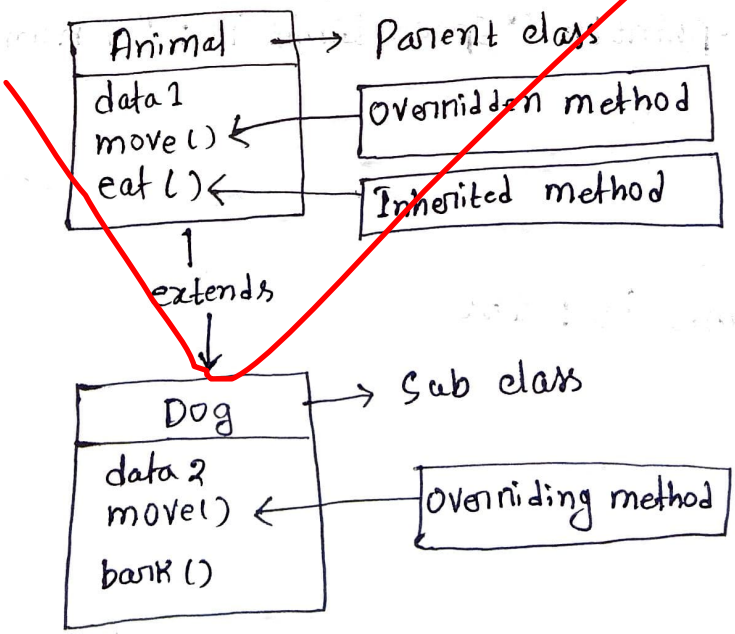
Semester: summer, 2020

Submission: 30-09-2020

Answer to the question 1(a)

Soln: If a method in a subclass has the same signature as a method in its superclass with the same return type, then the method is overridden. Justifying my answer in below:

If subclass has same method as declared in the parent class, it is known as method overriding in Java. It is a ~~fa~~ feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by ~~of~~ one of its ~~s~~ supper-classes or ~~parr~~ parent classes. When a method in a subclass has the same name, same parameters or signature, and same return type as a method in its super-class, then the method in the subclass is said to override the method in the supper-class.



```
// A simple Java program to ~~demonsto~~ demonstrate ~~method~~ method
// overniding in java.
Package ~~CSE~~ Mid_Term;
class carclass
{
  public int speedLimit()
  { return 100;
  }}
Class Ford extends CarClass
{
 public int speedlimit ()
{   return 150;
}
public static void main (String args[])
{ carclass obj = new Ford();
  int num = obj.speedLimit ();
System.out.println ("Speed Limit is: " + num);
}}
```

Output:

Speed Limit is : 150

Ans:

Answer to the question 1(b)

Part-1

Soln:

Class

A class in a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.

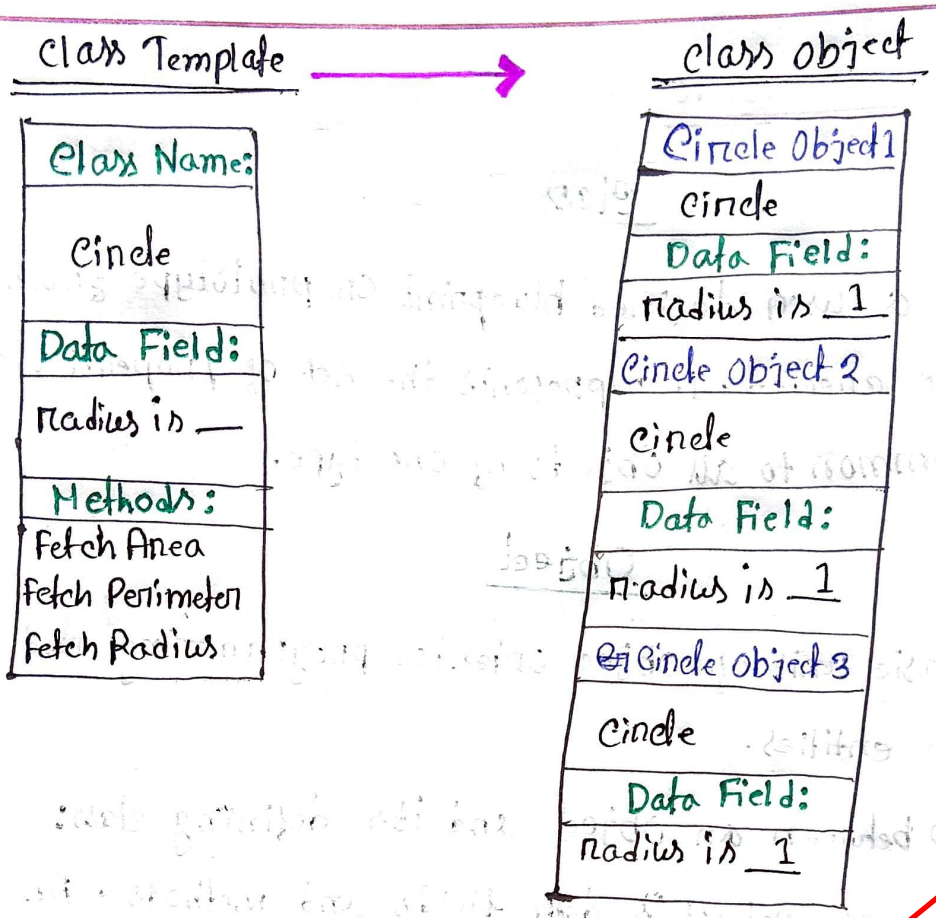
Object

It is a basic unit of object oriented programming and represents the real life entities.

Relationship between an object and its defining class:

class define an object's data fields and methods. An object is known as an instance of class. Such that object = instance.

To wrap up, the relationship of class to object is analogous to a chocolate chip cookie recipe to the cookies themselves. As we can make as many chocolate chip cookies we want from a single recipe, we can also make as many objects we want from a single class. Classes in Java use variables to design data fields. Classes ~~also~~ also use methods to design actions. Classes have ~~spe~~ ~~special~~ ~~special~~ ~~type~~ method of special type, called a constructor, which are invoked to make new objects.



We can make as many objects as we want from a class.

```
// A simple java program.
Package ~~Mid_Term~~; Mid_Term;
Public class circle {
double radius = 1;
circle ( ){ }
Circle ( double newRadius) {
radius = newRadius ; }
double FetchArea ( ) }
return radius * radius * Math.PI ; }
double fetchperimeter() {
return 2 * radius * Math.PI ; }
double setRadius (double newRadius) {
radius = newRadius ; } }
```

Part-2

"Bitwise operators" use to perform manipulation of individual bits of a number. They can be used with any of the integral types (char, short, int, etc). The "bitwise operators" are deals with the binary values, so it cannot handle decimal places. So, it could not be applied on double type data. Hence, the bitwise operators are used to check, set, or shift the bit values. So, it could not be used on double type as well as float and boolean type.

```
// A simple java program to illustrate bitwise operator.
public class operators {
public static void main(String[] args){
 int a = 5;   // here we can use char
 int b = 7;   // but double float not (we can't use it)
// bitwise and
// 0101 & 0111 = 0101 = 5
System.out.println("a & b = " +(a & b));
// bitwise or 0101 | 0111 = 0111 = 7
System.out.println("a|b= " +(a|b));
}}
```

Output: a & b = 5

a | b = 7

Ans:

Answer to the question 2(a)

Soln: C++ ~~spu~~ supports multiple inheritance but and java doesn't support it. But by using interface, we can support the functionality of multiple inheritance. Interface looks like a class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract. Also, the variables declared in an interface are public, static & final by default. Java doesn't ~~support~~ allow multiple inheritance to avoid the ambiguity caused by it. To overcome this problem interface is use. An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
interface <interface name> {
// declear constant fields
// declear methods that abstract
// by default
}
```

```
// A simple java program to demonstrate multiple inheritance
//through default methods.
interface PI1{
//default method
default void show(){
System.out.println("Default PI1"); }}
interface ~~P2~~ PI2{
// default method
default void show (){
System.out. ~~pnit~~ println ("Default PI2"); }}
class TestClass implements PI1, PI2 {
~~pu~~ // Overriding default show method
public void show(){
PI1.supper.show(); // supper keyword to call the show
PI2.super.show(); }
public static void main (string ~~argr~~ args[]){
TestClass d = new TestClass ();
d.show (); }}
```

Output:

Default PI 1

Default PI 2

Answer to the question 2(b)

Soln: My first name: Syeda → A
My last name: Ibnat → T
} last characters

```
// A java program to print last character of my first and
// last name. (Alplabet A and T)

~~Package Mid_Term~~
Package Mid_Term;
impont java.util.Scanner;
~~public class character~~
public class Pattern3{
static char ch;
// Function to print the letter A
static void a(){
System.out.printf("\n");
for(int i=0; i<0; i<8 ; i++){
 System.out.printf(" ");
for(int a=0; a<8; a++){
if (i==0 && (a==0 || a==7))
System.out.printf(" ");
else if (a<2 || a>5){
System.out.printf("%c", ch);}
```

```
else if (i<2 || (i>3 && i<5)) {
System.out.printf("%c", ch); }
else
System.out.printf(" "); }
System.out.printf("\n"); }}
//Function to print the letter T
static void t()
{ System.out.printf("\n");
 for (int i=0; i<8; i++) {
System.out.printf(" ");
for (int t=0; t<8; t++){
if (i<2 && t<8) {
System.out.printf("%c", ch); }
if (i>1 && t<3)
System.out.printf(" ");
if (i>1 && t>2 && t<5) {
System.out.printf("%c", ch); }
if (i>1 && t>4)
System.out.print(" "); }
System.out.printf("\n"); }}
```

```
// Function to print ~~each~~ ~~character~~ the required pattern.
Static void printpattern (char[] str) {
~~int~~ ~~int=0~~
int in=0;
While (in < str.length){
char ch=str [in];
if (ch<95)
ch= (char) (ch+32);
Switch (ch){
Case 'a':
a();
break;
case 't':
t();
break; }
in++; }}
// Driver ~~class~~ code
public static void main (String[] args){
Scanner Sc = new Scanner (System.in);
System.out.println ("Enter the value of n:");
int n= Sc.nextInt();
ch = '#';
char [] str = " A T ".toCharArray();
printpattern(str);}}
```

Output:

Enter the value of n: 6

```
# # # # # #
# # # # # # # #
# #
# #
# # # # # # # #
# #
# #
# #
```

```
# # # # # # # #
# # # # # # #
# #
# #
# #
# #
# #
# #
```

Answer to the question 3(a)

Sol^n: When a Java class extends another, we call it a subclass. And with method overriding, inheriting classes may ~~takes~~ takes how we exped a class type to behave. Generally, when a subclass extends another class, it inherits the behavior of the superclass. There has some feature in java programming where some methods must be override in sub classes and some method can't override in subclass.

~~One of the examples~~ where method

~~One of~~ Interface is one of the features where method overriding is mandatory. To implement an interface, ~~we hav~~ methods override is must. For example:

Take an interface Identifiable<T extends Serializable>{

T getId();

}

T represents the type of the class that will be used for the id. So, if we use this interface in a database application, T may have the type Integer, for example. Another notable thing is that T is ~~serializable~~.

We can also overridde access-modifiers. The access modifier for an overriding method can allow more, but not less, access than the overridden method.

On the other hand private method and static method and final method can't be overriden. Private methods cannot be overridden as they are bonded during compile time.

// Java program to demonstrate the real scenario of Java method overriding

Class ~~Bok~~ Bank{

int getRateOfInterest() {return 0;}

}

```
class SBI extends Bank{
in getRateOfInterest(){ return 8;}}
class ICIC extends Bank{
in getRateOfInterest(){return 7;}}
class AXIS extends Bank{
int getRateOfInterest(){ return 9;}}
class Test 2{
Public static void main(string args[]){
SBI s=new SBI();
ICIC i=new ICICI();
AXIS a=new AXIS();
System.out.Println("SBI Rate of Interest:"+s.getRateOfInterest());
System.out.Println("ICICI: Rate of Interest:"+i.getRateOfInterest
                                                    ());
System.out.Println("AXIS Rate of Interest:"+a.getRateOf
                                                    Interest());
}
}
```

Answer to the question 3(b)

Soln: The class will get the method from the first interface that is inherited by the class. And method can not be overniden in interface since inheritance supports only methods declaration not method body.

If a class implements two interfaces that declare the ~~same method~~, ~~then~~ default methods with some name, then the same method will be used by clients of either interface. The methods that implement an interface must be ~~declared~~ declared public. Here is a small example class that implements the callback interface.

```
interface callback {
 void callback (int param); }
class client implements callback {
//implement callback's interface
Pablic void Callback (int p) {
System.out.println ("Callback called with "+p); } }
```

If I want to overnide the method in subclass ~~which~~ then static method will be use. And it is not possible ~~in~~ to overnide in this situation.

ID: 17183103020

Similar to Default Method in Interface, the static method in an interface can be defined in the interface, but these methods cannot be overridden in implementation classes.

```
// Java program to demonstrate static method in Interface.
interface NewInterface {
Static void hello()
{ System.out.println(" hello, New static method here ");}
void OverridMethod (string Str); }
public class InterfaceDemo implements NewInterface {
public static void main (string [] args)
```

Check with your answer:

1.

(a) Method Overriding. Example with explanation

(b) can't applied for double type. explanation with relevant example

2.

(a) using interface feature. Give a suitable example

(b) are you print last to character of your first & last name

3.

(a) example with function overloading limitation such as final, static etc.

(b) some cases it is possible and some cases it is not possible.

see chapter-9, page211 for details