Name = Abdullah Abid

ID = 16453

Paper = java

Program = Software Engineering

Section = "B"

Date = 29/6/2020

Semester = 2nd

Scanned with CamScanner

Name= Abdullah Abid
ID= 16453

Question (1)

Answer-

The use of access modifiers goes to the core concept of encapsulation. Aka'data hiding in Object-oriented development. Variables should never be public. That is the whole point of Private & Protected modifiers on them to prevent direct access to the variables themselves.
You provide methode to manipulate varibles. A method has to be Public in order to allow other Programmers (or class) access to that data. But by using methods, you can control how a variable is manipulated: which is the entire point b/c

Scanned with CamScanner

Nam= Abdullah Abid
ID= 16453

you have hidden the details of the varible behind the method.
This is the fundamental concept of oops encapsulation.
The Protected modifier only allow for access to the varible & or method by subclasses & other classes in a Package. Typically, Protected is used to Prevent others form access the data except in controlled ways. Kind of like an "invited guest" Status versus anyone on the Street just wanding by.
Private is the most restrictive. only the class itself can access Private values. This is useful where you want to Prevent subclasses from modifying variables except through the

Scanned with CamScanner

Name = Abdullah Abid
ID = 16453
Page (3)

Controls provided & deny that ability to everyone else. This is the ultimate encapsulation. Private methods are used to provide internal functionality that you never need or want to expose to anyone else.
Very common in framework implementations.

Default is not commonly used generally. Public is the rule for methods - you want others to use.
Private is the default for class-level variables & for variables you don't want subclasses to directly access.
Protected is typically used within related class to internal access to data but keep it away from outside users. Protected methods are used for internal

Name = Abdullah Abid
ID = 16453

class behaviors that are needed to be shared or ~~[illegible]~~ leveraged within a class or family of classes but not exposed for anyone else to see.
Private methods are used for pure internal behaviors to a class that will never be shared outside itself. not even to its children

Part (b)

```
// Animal.java file.
// public class.
Public class Animal // Name of class
{
 Public int legcount; // Public variable.
Public void display()
{
 System.out.println("I am an animal);
 system.out.println("I have 4 legs);
}
 }
Public static main
```

Name = Abdullah Abid
ID = 16453


```
}
Public Static void main(String args[]);
Animal = New animal();
animal.leg count = 4;
animal.display();
}
}
```

Out Put:

I am an animal.
I have 4 legs.

Explination.

Here,

- The public class "Animal" is accessed from the main class
- The Public variabl "legCount" is accessed from the main class.
- The Public method "display()" is accessed from the main class.

Question (2)

Answer

Public Access =

When methods, variable; classes, & So on are

Name = Abdullah Abid
ID = 16453

are declared "Public", then we can access them from anywhere. The Public access modifier has no scope restriction.

Protected Access Modifier

When methods & data members are declared "Protected", we can access them within the same Package as well as from Subclasses.

For example

```
class an Animal
{
Protected void display()
{
System.out.Println("I am an animal);
}
}
class Dog extend animal
{
Public static void main(String args[]);

Dog Dog = New Dog();
dog.display();
}
```

Name = Abdullah Abid.
ID = 16453

}

Output

I am an animal

Explaintion => In this example, we have a protected method named "display()" inside the "Animal1" class.
The "Animal1" class is inherited by the "Dog" class. Since Protected methods can be accessed from the child classes, we are able to access the method of "Animal1" class from the "Dog" class

We cannot declare classes or interfaces Protected in java.

Question No(3)

Answer.
Next Page.

Scanned with CamScanner

Name = Abdullah Abid
ID = 16453

- For method overriding (So runtime Polymorphism can be achived)
- For Code Reusability.

Terms used in inheritance:
class. A class is group of objects which have common properties.
It is a template or blueprint from which objects are created.
Subclass or child class is a class which inherits the other class.
It is also called derived class or extended class or child class.
Reusability => As the name specifies, reusability is a mechanism which facilitates you to reuse the fields & methods of the existing class when you create a new class.

```
class subclass-name extends superclass-name
{
  methods & fields
}
```

Name = Abdullah Abid
ID = 16453

Part (b)

```
Class Animal
{
Public void eat()
{
System.out.Println(". I can eat");
}
Public void Sleep()
{
System.out.println("I can Sleep);
}
}
Class Dog extends Animal
{
Public void bark()
{
System.out.println("I can bark");
}
}
Class Main
{
Public Static void main(string args[]):
Dog Dog = New Dog();
dog1. eat();
dog2. sleep();
```

Scanned with CamScanner

Name = Abdullah Abid
ID = 16453


```
dog1 . bark();
}
}
```

Output =

```
I can eat
I can sleep
I can bark.
```

Explination = Here, we have inherited a subclass "Dog" from superclass "Animal". The Dog class inherits the methods "eat" & "sleep" from the "Animal" class. Hence, objects of the "Dog" class can access the members of both the "Dog" class the "Animal" class.

Scanned with CamScanner

Name = Abdullah Abid.
ID = 16453

Question No (3)(4)

Answer

Polymorphism = Polymorphism is the ability of an object to take on many forms. The most common use of Polymorphism in OOPs occurs when a parent class reference is used to refer to a child class object. It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. once declared, the type of a reference variable can not be changed.
A reference variable can refer to any object of its declared type or any subtype of its declared type.

Name = Abdullah Abid
ID = 16453

A reference variable can be declared as a class or interface type.
For example:-
Public interface vegetarin{}
Public class Animal{}
Public class Deer extends Animal implements vegetarin{}
Now, the Deer class is ~~consio~~ considered to be polymorphic since this has multiple inheritence Following are true for the above example

- A Deer IS-A animal.
- A. Deer IS-A object.
- A Deer IS-A Deer.

When we apply the reference variable facts to a Deer object reference. the following declaration are legal.
e.g

```
Deer d = new Deer();
object o = d;
vegetratain v = d;
Animal a = d;
```

Scanned with CamScanner

Name = Abdullah Abid
ID = 16453

Part (b)

```
File nam : Employee.java
Public class employee
{
Private String name;
Private String address;
~~Pro~~
Private int number;
Public Employee (string name,
String Address, int );
System.out.Println ("constructing an employee");
This . name=name;
This . address = address;
This number = number;
}
~~Pou~~
Public void milcheck()
{System.out.Println("mailing").
System String tostring( )
{
return . name+" "+ adress
}
Public String setname( )
{
```

Name= Abidullah Abid
ID = 16453


```
return name;
}
Public String get Address()
{
  return address;
}
Public int get Number()
{
  return number;
}
}
```

Now Suppose we extend Employee class as:

```
Public class Salary extend employee
Private doubl Salary;
Public salary (string name, String address,)
 setsalary(salary);
}
Public void mailcheck()
{
 System.out.Println("within mailcheck")
 System.out.Println("Mailing=+":with salary"+ salary)
}
Public doubl getSalary()
{
```

Name = Abdullah Abid
ID = 16453


```
Salary = New salary;
}
}
Public doubl compute pay ();
{
System.out.println("computepay");
return Salary/52;
}
}
~~Public Static void cto~~
Public class virtualDemo
{
Public Static void main(String args[]);
Salary S = new Salary();
System.out.println();
S.mailcheck();
System.out.Println();
e.mailcheck();
}
}
```

Output = Constructing an Employee.
Call mailcheck usin
Salary refer within mailcheck of
Salary class.
Call Mailcheck using employee

Name = Abdullah Abid
ID = 16453

refer.
within mailcheck of Salary
class

Explination:

Here, we instantiate two Salary objects one using a Salary reference s, & the other using an employee reference e.
While invoking s.mailcheck(); the compiler sees mailcheck() in Salary class at compile time, & the JVM invokes mailcheck() in the Salary class at run time.

Name = Abdullah Abid
ID = 16453

Question No (5)

Answer

Java Abstract classes & Methods Data abstraction is the Process of hiding certain details & showing only essential information to the user.

Abstraction can be achieved with either abstract classes or interfaces. (~~which your~~ ~~will~~

The abstract Keyword is a non-access modifier, used for classes & methods.

Abstract class: Abstract class is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Abstract method: They can only be used in an abstract class, & it does not

Name = Abdullah Abid
ID = 16453

not have a body.
The body is provided by
the subclass (inherited from).
An abstract class can
have both abstract &
regular methods
~~For example:~~
~~abstract class~~

Rules of Abstract Method

~ If a class is using
an abstract method they
must be true. declared abstract
~~this mean~~
The opposite cannot be true.
~ Abstract method do not have
an implementation. it only
has method signature.
~ If a regular class extends
an abstract class, then that
class must implement all
the abstract method of the
abstract parent.

Scanned with CamScanner

Nam= Abdullah Abid
ID= 16453

Part(b)

```
Abstract class Shape.
{
String color; // These are
abstract methods :abstract double area();
Public abstract String toString();
// Abstract class can be have
constructor.
Public Shape(string color)
{
System.out.Println(" Shape constructor called");
this.color= color;
}
// This is a concrete method
Public String getColor()
{
return color;
}
}
class Circle extend Shape
{
double radius;
Public Circle(string color.double radius)
{
```

Scanned with CamScanner

Name = Abdullah
ID = 16453

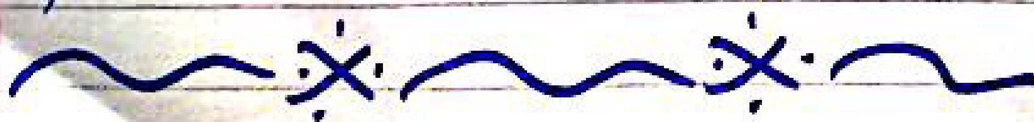

```
// calling Shape constructor super (color);
System.out.println("(Circle constructor called");
this. radius = radius;
double area()
{
return Math.PI * Math.Pow(radius, 2);
}
Public String toString()
{
return "Circle color is" + super.color +
" and area is: " + area();
}
}
class Rectangle extend Shape
{
double length;
double width;
Public Rectangle
{
System.out.println("Rectangle constructor called");
this. length = length;
this . width = width;
}
double area()
{
```

Scanned with CamScanner

Name = Abdullah Abid
ID= 16453


```
return length * width;
}
Public String tostring()
{
return "Rectangel color is "+ super.color+
 "and area is: "+ area();
}
}
Public class Test
{
public static void main(string arg[])
{
Shapes1 = New circle ("Red. 3.3);
Shapes 2 = New Rectangle("Green". 3.6);

System. out. println(s1. tostring());
System. out. println(s2. tostring());
}
}
```



The End.

Scanned with CamScanner