Sanjana Kumbhar

24/11/24 Final Java Assignment

Que.4 →

Q1- Which type of java version you used

Ans- ↳ java - JDK - 21
latest version 23

Q2- Types of variables in Java

Ans- Variables is a field that stores data value which used during program execu"

Types, →

1) local variable - declared in methods, constructor or blocks. (scope)
. Access modifiers cannot be used for Local var
. Local var are implemented at stack level internaly. Cannot have default value need to assign before use"

2) Instance variable - declared in class but outside a method, constr or any block (heap memory)
. Inst. var are created when obj is created with 'new' & destroy when obj is destroyed. It holds values for particular obj. we can give access modifier to it
. It has default values (boolean=false, int=0)
. within class access it directly, outside class "objname.varnam" with name

3) static / class variable -
declare with 'static keyword' in class (outside method, constructer or block)
It store in static memory. It is created when program starts & end with program ends, It has default values can access by → classname.varname
if it is → ① public static final NAME (upper case may)
② public static name.

- diff. betn static & final →
  - ↳ static is used to represent class member. It is final is like const variable
  - ↳ static var are class members and can be reinitialized while final variables are constants
  - ↳ static var are same for all instance

```
class Testing () {
    int myvar1;  // Instance var
    static int data=30; // static / class var
    public void cal (){
        data = 40;
        int a = 10;  // local var
        sysout(a);  ——→ 10
        sysout(myvar);  ——→ 0
        sysout(Testing.data);  ——→ 40
        sysout(data)  ——→ 40
    }
main() {
    Testing T1 = new Testing ();
    Sysout (T1.myvar)  ——→ 0
    syout (Testing.data)  ——→ 40
    sysout(a)  ——→ Error
    sysout(data)  ——→ 40
    Testing T2 = new Testing()
    sysout (T2.data)  ——→ 40.
```

modify static var.

M T W T F S S

Page No.: YOUVA

Date:

(3.3) differentiate method overloading & overriding

Method overloading	Method Overriding
① compile time polymorphi.	① Run time Polymorphism
② It occurs within class	② It performed in 2 classes within inheritance relan
③ may or may not require inheritance	③ needs inheritance
④ methods have same name & diff signature (i/p para)	④ methods have same name & same signature
⑤ private & final method can be overloaded.	⑤ public & final method cannot overridden

```
class Moverload (){
static int add (int a, int b){
    return a+b;
}
static int add(int a, int b, int c){
    return a+b+c;
}
main (string args[]){
sysout (add (4,6));
sysout (add (1,2,3));
}
```

```
class Animal () {
    void eat () {
        sysout ("Animal-eat");
    }
class cat extends animal (){
    void eat () {
        sysout ("cat-eat");
    }
    void anieat ()
        super.eat()}
}
class Moverride () {
    main () {
cat c1 = new cat ();
animal a1 = new animal()
a1.eat(); ——→ animal eat
c1.eat (); ——→ cat eat
animal a2 = new cat();
a2.eat ——→ cat eat
c1 / a1.anieat ——→ animal eat
}
```

4) Explain constructer & types constructer

It is special method that is used to initialize obj. It is called when an obj of class is created. It can be used to set initial values for obj attributes or instance variables

Types ⟶

① default constructer -
constructer that has no parameters. It is invisible

② Parameterized constructer -
constructer that has parameters. If we want to initialize fields of class with our values

③ copy constructer -
copy constructors copy constructer is passed with another obj which copies the data available from the passed obj to newly created obj

```
class Test{
    int val1;
    string name;

Test(){
    val1 = 1;
    name = "sam";
}

Test(int n, string s){
    this.val1 = n;
    this.name = s;
}
```

```
Test(Test obj2){
this.val = obj2.val;
this.name = obj2.name;}

main(){
Test t1 = new Test();
  ↳ val1 = 1, name = sam

Test t2 = new Test(5, "Tom");
  ↳ val1 = 5  name = Tom

Test t3 = new Test(t2);
  ↳ val1 = 5  name = Tom
```

95) constructor chaining →

it is process of calling one constructor from another constructor with respect to current obj & make code more readable

2 methods

① within same class -

It can be done using this() keyword for constructor in same class

② from base class - using super() keyword to call constr. from base class

```
class Test {
①  Test(){
      this(5);
      sysout("default constr");
   }
   Test(int x){
      this(5,15);
      sysout(x);
   }
   Test(int x, y){
      sysout(x+y);
   main(){
Test t1 = new Test();
```

o/p

20
5
default constr()

Q f) Explain concept of priority queue →

Priority queue is used when objs are supposed to be processed based on priority

queue — FIFO

but in priority queue - out is based on priority

- priority queue is based on priority heap

ex:

Insert (c)	c
—11— (o)	co
insert (D)	coD
remove max() — (o) —	cD
insert (I)	cDI
—11— (N)	cDIN
insert (G)	cDING
remove max() — (N) —	cDIG

Q → Explain multithreading & synchronized in java

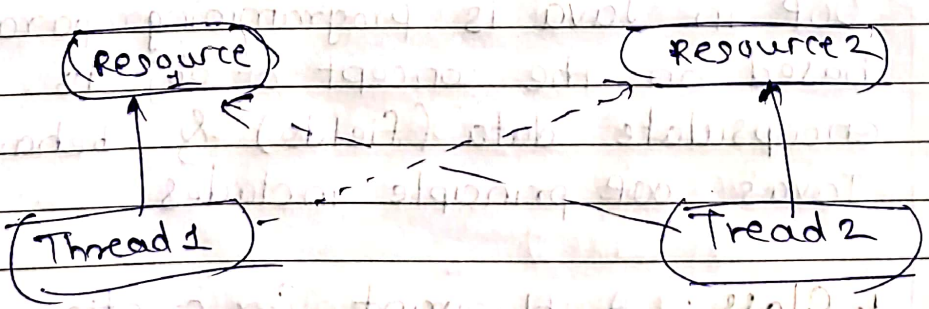
↳ multithreading → It allow concurrent execution of two or more parts of program for max utilized of CPU each part of such program is called thread

2 mechanism
① extending Thread class
② implementing runnable Interface

synchroniza^n —
is capability to control access of multiple threads to any shared resource.
It is better option where we want to allow only one thread to access resource

types → process synchroniza^n
→ Thread synchroniza^n
① mutual exclusive
↳ i) synchronized method
2) synchronized block
3) static synchroniza^n
② coopera^n (Inter thread communica^n)

Q.) deadlock situa^n in java
It is part of multithreading. It occurs in a situa^n when a thread is waiting for an obj lock, that is acquired by another thread & 2nd thread is waiting for an obj lock that is acquired by 1st thread. Since, both threads are waiting for each other to release lock.



Here T1 acquire R1 & T2 acquire R2 but T1 requires R2 to complete its process & T2 requires R1 to complete its process so T1 & T2 cannot getting R1 / R2 so here deadlock happened

Q5) Explain recursion using one example

Recursion is technique of making function call itself. It provides a way to break complicated problems into simple problems.

```
Public class Main {
    Public static void main (string [] args) {
        int result = sum(10);
        System.out.println(resut);
    }
    Public static int sum (int k) {
        if (k > 0) {
            return k + sm(k-1);
        } eles {
            return 0;
        }
    }
}
```

Explain object oriented programming in Java

OOP in Java is programming paradigm based on the concept of objects, which encapsulate data (fields) & behavior (methods) Java's OOP principle includes

1. Class : A blueprint for creating objects

Example :

```
class Car {
    String brand;
    void drive() {
        System.out.println("Driving" + brand);
    }
}
```

2. Object: An instance of class.

Example:

```
Car car = new Car();
car.brand = "Toyota";
car.drive();
```

3. Encapsulation: Restricting access to fields & methods using access modifiers (private, public, etc).

Example:

```
class Car {
    private String brand;
    public void setBrand(String brand) {
        this.brand = brand;
    }
    public String getBrand() {
        return brand;
    }
}
```

4. Inheritance: Enabling a class to inherit fields & methods from another class.

Example :

```
Class Vehicle {
    void start() {
        system.out.println("Vehicle started");
    }
}
Class Car extends Vehicles {
    void honk() {
        System.out.println("Car honks");
    }
}
```

5. Polymorphism: Allowing methods to have different implementations (overlooking & overriding)

Example

```
Class vehicle {
    void run() {
        System.out.println("Vehicle is running");
    }
}
class car extends Vehicle {
    @override
    void run() {
        system.out.println("Car is running");
    }
}
```

6. Abstraction: Hiding implementation details using abstract classes or interfaces.

Example:

```
abstract class Vehicle {
    abstract void run();
}
class car extends Vehicle {
    void run() {
        system.out.println("Car is running");
    }
}
```

Q Explain jdk jre & jvm

1. JVM (Java Virtual Machine):
   - It is a runtime enviornment that executes Java bytecode.
   - Converts bytecode to machine code (via Just-In time complilation)
   - Provides platform independence & manages memory though garbage collection.
   - Example: When you run a Java ~~cod~~ program (java Myclass), the JVM executes it.

2. JRE (Java Runtime Enviornment)
   - Provides the libraries & JVM required to run Java applications
   - Include JVM, core classes & supporting files
   - Does not include ~~imy~~ developement tools like comptilers.
   - Example: To run a .class file, you only need the JRE.

3. JDK (Java Developement kit):
A software developement kit required to develop Java application. It includes tools like compiler (javac), Java runtime environment (JRE), & debugger.

Heirarchy:
JDK > JRE > JVM

@ Explain use of lambda expression

Lambda expressions in Java provide a consise way to represent anonymous functions. They are primarily used to simplify code, especially in functional programming.

Key Uses:

1. Functional Interfaces:
Used with interfaces having a single abstract method (eg. Runnable, Comparator).

(Parameters) -> { body }

2. Stream API:
Simplifies operations like filtering, mapping & reducing collections.

```
List<Integer> nubers = Arrays.aslist(1,2,3,4);
numbers.stream().filter(n -> n%2==0).
forEach(System.out::println);
```

3. Event handling :
Reduces boilerplate for event listeners.

```
button.addActionListener(e -> System.out.println
("Button clicked"));
```

4. Improved readability :
Replaces verbose anonymous inner classes.

Example :

```
Runnable r = () -> System.out.println("
Lambda expression example");
new Thread(r).start();
```