SixthTask

```
// Name : Hanif Yogatama
// Date : Monday , 29 June 2020
```

Questions

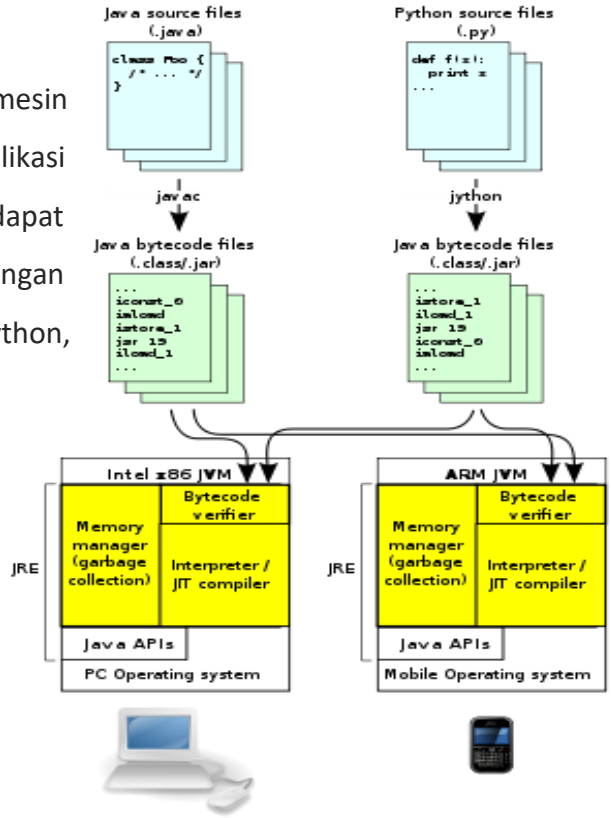
1. Cari tau apa itu JVM (Java Virtual Machine) ?
2. Cari tau apa itu :

    a. Class
    b. Object
    c. Encapsulation
    d. Access Modifier
    e. Inheritance
    f. Constructor
    g. Polymorphism
    h. Overloading
    i. Overriding

3. Cari tau apa itu Destructor ?
4. Di Java atau Kotlin apakah ada Destructor ? jika ada / tidak jelaskan !
5. Bahasa pemrograman apa saja yang ada Destructornya ?
6. Fungsi dan cara kerja Garbage Collection ?

Answer 1

Java Virtual Machine (JVM) adalah mesin virtual yang digunakan untuk menjalankan aplikasi berbasis Java serta , selain itu JVM juga dapat digunakan oleh bahasa pemrogrman lain dengan mengubahnya menjadi “java bytecode” seperti Jython, Groovy, JRuby .



Answer 2

- **Class ,** adalah suatu rencana atau rancangan yang mendefinisikan `variabel` dan menciptakan suatu `instant` dari `object`.
- **Object,** adalah hasil dari representasi yang dibuat pada sebuah `class`, singkatnya objek juga bisa disebut cetakan dari sebuah `class`.

```
class Lamp {
// contoh class dan method
    private var isOn: Boolean = false
    fun turnOn() {
        isOn = true
    }

    fun turnOff() {
        isOn = false
    }

    fun displayLightStatus(lamp: String) {
        if (isOn == true)
            println("$lamp lamp is on.")
        else
            println("$lamp lamp is off.")
    }
}

fun main(args: Array<String>) {

    val l1 = Lamp()
    val l2 = Lamp()

    l1.turnOn()
    l2.turnOff()

    l1.displayLightStatus("l1")
    l2.displayLightStatus("l2")
}
```

- **Encapsulation,** adalah sebuah proses penyatuan attribute bersama method menjadi satu unit data yang disebut member, untuk menjaga suatu proses program agar tidak dapat diakses secara sembarangan oleh program lain. Jika ingin mengaksesnya biasanya mengguanakn access modifier sebagai penentu dari mana saja member tersebut dapat diakses.

```
Class Anak : Bapak() {
    private val nama : String = “leonardo”
    private val umur : Int = 30
    private val marga : String = “sihombing”

    override fun ambisius() {
        super.ambisius()
     }
}
```

- **Access Modifier,** adalah sebuah hak akses yang diberikan kepada sebuah variable / method / class dengan tujuan menjaga integritas dari data tersebut ketika ingin diakses object lain. Dalam kotlin ada 4 yakni Public, Private, Internal, Protected.

```
open class Student  {
   var nama = "purwoto"
   private var umur = 29
   internal var jeniskelamin = "pria"
   protected fun e(){}
}
```

- **Inheritance,** adalah konsep pemrograman dimana sebuah class dapat menurunkan/mewariskan property dan method yang dimilikinya kepada class lain

```
open class Bapak {
   open fun introvet(){

   }
   open fun pelupa(){

   }
   open fun(){

   }
}
```

```
open class Anak : Bapak {
   overrade fun introvet(){
     super.introvet()

   }
   overrade fun pelupa(){
     super.pelupa()

   }
   overrade fun diabetes(){
     super.diabetes()

   }
}
```

- **Constructor,** adalah proses menginisialisasi property dalam mempersiapkan object.

```
fun main (args: Array<String>) {

    val seseorang:Manusia= Manusia("hanif","Kalibata City")
    println("Nama : ${seseorang.nama} dan umur : ${seseorang.alamat}")

}
```

- **Polymorphism,** adalah sebuah prinsip dimana class dapat memiliki beberapa bentuk (parameter , tipe data) method yang berbeda meskipun dengan nama yang sama.

  Static Polymorphism , menggunakan method overloading

```
Class lingkaran  {
    fun luas(r: Float): Float {
        return (Math.PI * r * r).toFloat()
    }

    fun luas(d: Double): Double {
        return (1/4 * Math.PI * d)
    }
}
```

Static Polymorphism , menggunakan method overriding

```
open class BangunDatar {
   open fun luas(): Float{
     println("menghitung luas bangun datar ")
     return 0F
   }

   open fun keliling(): FloatP{
     println("menghitung keliling bangun datar ")
     return 0F
   }
}
```

```
class Lingkaran(var r:Int) : BanguDatar() {
   overrade fun luas(): Float{
      return (Math.PI * r * r).toFloat)
   }
    overrade fun keliling(): Float{
      return (2 * Math.PI * r).toFloat)
   }
}
```

```
class Persegi(var sisi:Int) : BanguDatar()
{
   overrade fun luas(): Float{
      return (sisi * sisi).toFloat)
   }
    overrade fun keliling(): Float{
      return (sisi * 4).toFloat)
   }
}
```

```
fun main(){
    val banguDatar = BangunDatar()
    val persegi = Persegi(4)
    val lingkaran = Lingkaran(5)

    bangunDatar.luas()
    bangunDatar.keliling()

    println("Luas Persegi : ${persegi.luas()}")
    println("Keliling Persegi : ${persegi.keliling()}")
    println("Luas Lingkaran : ${lingkaran.luas()}")
    println("Keliling Lingkaran : ${lingkaran.keliling()}")
}
```

- **Overloading method,** merupakan suatu method yang memiliki kesamaan nama method namun dengan parameter berbeda
- **Overriding method,** merupakan method yang parrent class yang ditulis kembali oleh subclass pada sebuah program. Method ini memungkinkan subclass untuk merubah dan memodifikasi statement-statement pada method yang dimiliki oleh superclass.

Answer 3

Destructor adalah suatu method khusus yang akan dieksekusi saat objek dihapus dari memori.

Answer 4

Tidak memiliki destructor , karena di dalam Java atau Kotlin memiliki fitur manajemen memori sendiri yakni Garbage Collector yang fungsinya sudah mewakili destructor.

Answer 5

C++, PHP

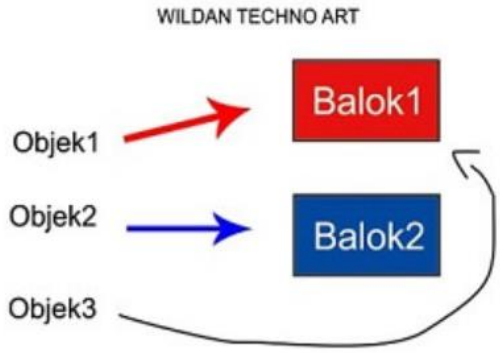
Answer 6

Garbage Collection merupakan salah satu mekanisme dari fitur JVM yang berfungsi untuk meningkatkan manajemen memori yakni dengan menghapus sebuah objek yang tidak diperlukan atau tidak direferensikan lagi.

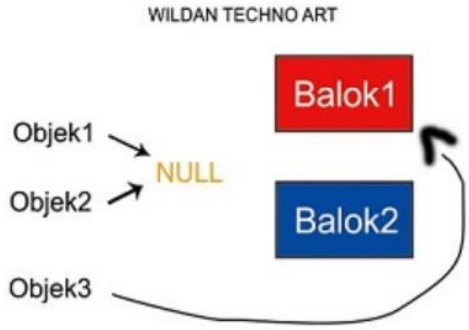
```java
package WILDAN_TECHNO_ART;

public class Balok {

    //Method ini akan ditampilkan jika Garbage Collection bekerja
    public void finalize(){
        System.out.println("Objek Yang Tidak Terpakai Sudah Dibersihkan:");
    }

    public static void main(String[] args){
        Runtime RT = Runtime.getRuntime();
        System.out.println("Jumlah Memori Awal: "+RT.totalMemory());

        Block objek1 = new Block(); //Block Satu
        Block objek2 = new Block(); //Block Dua
        Block objek3 = objek1; //Objek3 Mengembalikan Nilai Objek1
        objek1 = null;
        objek2 = null;
        System.out.println("Jumlah Memori Yeng Tersedia Sebelum di GC: "+RT.freeMemory());
        System.gc();
        System.out.println("==========================================");
        System.out.println("Jumlah Memori Yeng Tersedia Setelah di GC: "+RT.freeMemory());
    }
}
```

Cara Kerja

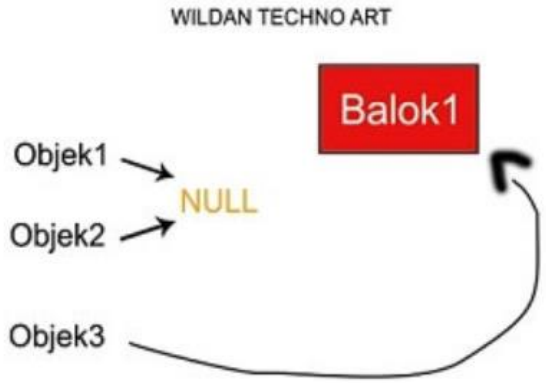
- Pada program tersebut, disana terdapat 3 buah objek atau variable dari Class Balok, objek1 dan objek2 diinisialisasikan kedalam Balok1 dan Balok2, dan objek3 mempunyai nilai kembali dari objek Balok1.



- Pada baris selanjutnya, objek1 dan objek2 kita ubah nilainnya menjadi *null* yang berarti kosong, sedangkan objek3 masih mengembalikan nilainya objek1.



- Karena pada Balok2 tidak ada yang direferensikan nilainya, maka secara otomatis, Balok2 tersebut akan di bawa oleh System Garbage Collection pada JVM.



- *System.gc()*, berfungsi untuk menjalankan Garbage Collection pada java. lalu Method *finalize()*, digunakan untuk memastikan bahwa objek telah bersih dan Garbage Collection telah bekerja, Di dalam metode *finalize()*, kita dapat menentukan tindakan yang harus dilakukan sebelum suatu objek hancur.

  Dan terakhir, kita menambahkan sebuah variable *Runtime*, yang digunakan untuk mengecek Total Memori serta Jumlah Memori yang tersedia sebelum dan sesudah di Garbage Collection