|  |  |
| --- | --- |
| INTERPRETER | COMPILER |
| Interpreter translates just one statement of the program at a time into machine code. | Compiler scans the entire program and translates the whole of it into machine code at once. |
| An interpreter takes very less time to analyze the source code. However, the overall time to execute the process is much slower. | A compiler takes a lot of time to analyze the source code. However, the overall time taken to execute the process is much faster. |
| An interpreter does not generate an intermediary code. Hence, an interpreter is highly efficient in terms of its memory. | A compiler always generates an intermediary object code. It will need further linking. Hence more memory is needed. |
| Keeps translating the program continuously till the first error is confronted. If any error is spotted, it stops working and hence debugging becomes easy. | A compiler generates the error message only after it scans the complete program and hence debugging is relatively harder while working with a compiler. |
| Interpreters are used by programming languages like Ruby and Python for example. | Compliers are used by programming languages like C and C++ for example. |
| It requires source code for later execution. | It does not require source code for later execution. |
| Due to interpreters being slow in executing the object code, it is preferred less. | Main advantage of compilers is it’s execution time. |

1.DIFFERENCE BETWEEN INTERPRETER AND COMPILER.?

2.tight coupling.

**Tight coupling** means classes and objects are dependent on one another. In general, tight coupling is usually not good because it reduces the flexibility and re-usability of the code while **Loose coupling** means reducing the dependencies of a class that uses the different class directly.

* The **tightly coupled object** is an object that needs to know about other objects and is usually highly dependent on each other's interfaces.
* Changing one object in a **tightly coupled application** often requires changes to a number of other objects.
* In the small applications, we can easily identify the changes and there is less chance to miss anything. But in large applications, these inter-dependencies are not always known by every programmer and there is a chance of overlooking changes.

class A {

   public int a = 0;

   public int getA() {

      System.out.println("getA() method");

      return a;

   }

   public void setA(int aa) {

      if(!(aa > 10))

         a = aa;

   }

}

public class B {

   public static void main(String[] args) {

      A aObject = new A();

      aObject.a = 100; // Not suppose to happen as defined by class A, this causes tight coupling.

      System.out.println("aObject.a value is: " + aObject.a);

   }

}

2.loose coupling

**Loose Coupling**

* **Loose coupling** is a design goal to reduce the inter-dependencies between components of a system with the goal of reducing the risk that changes in one component will require changes in any other component.
* **Loose coupling** is a much more generic concept intended to increase the flexibility of the system, make it more maintainable and makes the entire framework more stable.

class A {

   private int a = 0;

   public int getA() {

      System.out.println("getA() method");

      return a;

   }

   public void setA(int aa) {

      if(!(aa > 10))

         a = aa;

   }

}

public class B {

   public static void main(String[] args) {

      A aObject = new A();

      aObject.setA(100); // No way to set 'a' to such value as this method call will

                         // fail due to its enforced rule.

      System.out.println("aObject value is: " + aObject.getA());

   }

}

3.stack overflow.

public class SimpleExample {

public static void main(String args[]) {

a()

}

public static void a() {

int x = 0;

b();

}

public static void b() {

Car y = new Car();

c();

}

public static void c() {

float z = 0f;

System.out.println("Hello");

}

}

4.work on data types declaration. ?

**class** FirstProgram {

**public** **static** **void** main(String[] args) {

**int** x=1;

**float** test=1.2f;

**double** db=22.5d;

**char** ch='s';

String myname="sai";

**boolean** N1=**true**;

System.***out***.println("integer :" + x);

System.***out***.println("float value is : "+test);

System.***out***.println("double value is : "+db);

System.***out***.println("char value is : "+ch);

System.***out***.println("Strimg value is : "+myname);

System.***out***.println("boolean value is : "+N1);

}

}

5. simple code for

(1). if else

**import** java.util.Scanner;

**public** **class** SecondProgram {

**int** age=22;

**public** **static** **void** main(String[] args) {

Scanner Sc= **new** Scanner(System.***in***);

System.***out***.println("enter the value");

**int** x=Sc.nextInt();

**if**(x>=18) {

System.***out***.println(x+" is eligible for vote");

}

**else** {

System.***out***.println(x+"is not eligible for vote");

}

}

}

(2)for loop, while and do while

// forloop

**public** **class** ThirdProgram {

**public** **static** **void** main(String[] args) {

//forloop

**for**(**int** i=1;i<=10;i++){

System.***out***.println(i);

}

//while

**int** m=1;

**while**(m<=10){

System.***out***.println(m);

m++;

}

//dowhile

**int** k=1;

**do**{

System.***out***.println(k);

k++;

}**while**(k<=10);

}

}

6. (1) create parameterized constructor using variables.

**public** **class** FifthProgram {

String name;

**int** age;

**int** wt;

**public** FifthProgram(String name, **int** age, **int** wt) {

**this**.name = name;

**this**.age = age;

**this**.wt = wt;

System.***out***.println(name+" "+"your age is : "+age+" "+"and your weight is : "+wt);

}

**public** **static** **void** main(String[] args) {

FifthProgram s1=**new** FifthProgram("sai",23,65);

FifthProgram s2=**new** FifthProgram("raja",28,75);

FifthProgram s3=**new** FifthProgram("bhavna",18,50);

}

}

6 .(2)create zero param constructor and parametrized constructor.

// zero parametrized constructor and parameterized constructor and

**public** **class** SixProgram {

String name="ram";

**int** age;

**int** wt;

**public** SixProgram() {

**this**.name="sai";

System.***out***.println(name);

}

**public** SixProgram(String name, **int** age) {

**this**.name = name;

**this**.age = age;

System.***out***.println(name+ " your age is : "+age);

}

**public** SixProgram(String name, **int** age, **int** wt) {

**super**();

**this**.name = name;

**this**.age = age;

**this**.wt = wt;

System.***out***.println(name+ " your age is : "+age+" and your weight is : "+wt);

}

**public** **static** **void** main(String[] args) {

SixProgram s1=**new** SixProgram();

SixProgram s2=**new** SixProgram("raja",30);

SixProgram s3=**new** SixProgram("bhavna",18,65);

}

}

ARRAYS

We can write declaration of arrays two ways

1.datatype arr[]=new datatype[size(length)];

2. datatype arr[]={data1,data2,…..data n};

Limitations:

1: Homogeneous in nature.

2: size is fixed.

7. create an array with 10 elements and inside the loop.

**public** **class** Program {

**public** **static** **void** main(String[] args) {

int ar[]={1,2,3,4,5,6,7,8,9,10};

**for**(**int** i=1;i<=10;i++){

System.***out***.println(i);

}

}

}