**Types of variables**

**Division 1**

Based on type of value represented by a variable all variables are dived into 2 types.

1. Primitive variables
2. Reference variables

Primitive variables are used to represent primitive values

int x = 10;

Reference variables can be used to refer objects

Student s = new Student();

**Division 2**

Based on position of declaration and behavior all variables are divided into 3 types

1. Instance variables
2. Static variables
3. Local variables

**Instance variables**

* If the value of a variable is varied fromobject to object such type of variables are called instance variables.
* For every object a separate copy of instance variable will be created.
* Instance variables should be declared with in the class directly but outside of any method or block or constructor.
* Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly same as the scope of object.
* Instance variables will be stored in the heap memory as the part of object.

**Ex**

S1 s2 s3

Name: pallavi name: Rani Name: Gayatri ……………

R.No: 101 R.No: 102 R.No:103

Ex

class Test

{

int x = 10;

public static void main(String args[])

{

S.o.pln(x); CE: non static variable x cannot be referenced from a static context

Test t = new Test();

S.o.pln(t.x); ======10

}

Public void m1()

{

S.o.pln(x); ==========10

}

}

* We cannot access instance variables directly from static area but we can access by using object reference.
* But we can access instance variables directly from instance area
* For instance variables JVM will always provide default value and we are not required to perform initialization explicitly.

class Test

{

int x;

double d;

boolean b;

String s;

p.s.v.m(String args[])

{

Test t1 = new Test();

S.O.pln(t1.x); ------0

S.O.pln(t1.d); ------0.0

S.O.pln(t1.b); ------false

S.O.pln(t1.s); ------null

Instance variables also known as **object level variables** or **attributes.**

**Static variables**

*Declaration of static variable*

Static variable should be declared with in the class directly but outside of any method or block or constructor.

*When static method will be created when it will be destroyed*

Static variables will be created at the time of class loading and destroyed at the time of class unloading hence the scope of static variable is exactly same as scope of .class file.

When we execute the java file that is java .classfilename internally the following activities will be done.

1. Start JVM
2. Create and start main thread
3. Locate .class file (search where the file is available)
4. Load .class file
5. Execute main() method
6. Unload .class file
7. Terminate main thread
8. Shutdown JVM

Static variable will be created at 4th step; static variable will be destroyed at 6th step.

Static variables will be stored in method area.

We can access static variables either by object reference or by class name. But recommended to use class name.

Within the same class it is not required use class name and we can access directly.

**EX:**

class Test

{

static int x = 10;

public static void main(String args[])

{

Test t = new Test();

System.out.println(t.x);

System.out.println(Test.x);

System.out.println(x);

}

}

Output

10

10

10

We can access static variables directly from both instance and static areas.

For static variables JVM will provide default values and we are not required to perform initialization explicitly.

EX

class Test

{

static int x;

static double d;

static String s;

public static void main(String args[])

{

System.out.println(x);

System.out.println(d);

System.out.println(s);

}

}

Output

0

0.0

Null

Static variables are also known as class level variables or fields.

class Test

{

static int x=10;

int y = 20;

public static void main(String args[])

{

Test t1 = new Test();

t1.x = 888;

t1.y = 999;

Test t2 = new Test();

System.out.println(t2.x+"..."+t2.y);

}

}

output

888...20

**Local variables**

* Sometimes to meet temporary requirements of the programmer we can declare variables inside a method or block or constructor, such type of variables are called local variables or temporary variable or stack variables or automatic variables.
* Local variables will be stored inside stack memory.
* Local variables will be created while executing the block in which we declared it.
* Once block execution completes automatically local variable will be destroyed. Hence the scope of local variable is the block in which we declared it.
* For local variables JVM won’t provide default values compulsory we should perform initialization explicitly. Before using that variable that is if we are not using then it is not required to perform initialization

**EX 1:**

class Test

{

public static void main(String args[])

{

int x;

System.out.println("Hello"); // Here we are not using any x variable

}

}

**Output**

Hello

**EX 2**

class Test

{

public static void main(String args[])

{

int x;

System.out.println(x); // Here we are not using any x variable

}

}

**Output**

CE- variable x might not have been initialized.

Flow control

Flow control describes the order in which the statements will be executed at runtime

There are 3 types of flow control statements

1. Selection statements

1. if-else

2. switch()

1. Iterative statements
2. while
3. do-while
4. for
5. for-each
6. Transfer statements
7. break
8. continue
9. return
10. try-catch-finally
11. assert
12. **Selection statements**

1**. if-else**

Syantax

if(b)

{

action if b is true

}

else

{

action if b is false

}

The argument to the if statement should be Boolean type. By mistake if we are trying to provide any other type then we will get compile time error.

Case 1

class Test

{

public static void main(String args[])

{

int x = 0;

if(x)

{

System.out.println("hello");

}

else

{

System.out.println("Hi");

}

}

}

Output

incompatible types: int cannot be converted to Boolean

case 2

class Test

{

public static void main(String args[])

{

int x = 10;

if(x=20)

{

System.out.println("Hello");

}

else

{

System.out.println("Hi");

}

}

}

Output

incompatible types: int cannot be converted to Boolean

case 3

class Test

{

public static void main(String args[])

{

int x = 10;

if(x==20)

{

System.out.println("Hello");

}

else

{

System.out.println("Hi");

}

}

}

Output

Hi

Case 4

class Test

{

public static void main(String args[])

{

boolean b = true;

if(b=false)

{

System.out.println("Hello");

}

else

{

System.out.println("Hi");

}

}

}

Output

Hi

Case 5

class Test

{

public static void main(String args[])

{

boolean b = false;

if(b==false)

{

System.out.println("Hello");

}

else

{

System.out.println("Hi");

}

}

}

Output

Hello

Case 1

class Test

{

public static void main(String args[])

{

if(true)

System.out.println("Hello");

}

}

Output

Hello

Case 2

class Test

{

public static void main(String args[])

{

if(true);

System.out.println("Hello");

}

}

Output

Hello

Else part and curly brases are optional. Without curly braces only one statement is allowed. under if which should not be declarative statement.

class Test

{

public static void main(String args[])

{

if(true)

int x =10;

}

}

Output

variable declaration not allowed here

class Test

{

public static void main(String args[])

{

if(true)

{

int x =10;

}

}

}

Output

Nothing will be displayed

semicolon is a valid java statement which is also known as empty statement.

Note:

There is no dangling else problem in java. Every else is matched to the nearest if statement.

Switch

If several options are available then it is not recommended to use nested if else. Because it reduces readability. To handle this requirement we should go for switch statement

Syntax:

Swich(x)

{

case 1:

Action 1;

Break;

case 2:

Action 2;

Break;

‘

‘

‘

case n:

Action n;

Break;

default:

Default action

}

The allowed argument types for the switch statement are byte, short, int, char until 1.4 version. But from 1.5 version onwards corresponding wrapper classes and enum classes are also allowed. From 1.7 version onwards string type is also allowed.

Curly braces are mandatory except switch everywhere curly braces are optional

Both case and default are optional. That is an empty switch statement is a valid java syntax.

Inside a switch every staten

**Static variables**

* If the value of a variable is not varied from object to object then it is not recommended to declare variable as instance variable.
* We have to declare such type of variables at class level by using static modifier
* In the case of instance variables for every object a separate copy will be created but in the case of static variables a single copy will be created at class level and shared by every object of the class.
* Static variables should be declared with in the class directly but outside of any method or block or constructor.
* Static variables will be created at the time of class loading and destroyed at the time of class unloading hence scope of static variable is exactly same as scope of .class file

**Ex**

S1 s2 s3

Name: pallavi name: Rani Name: Gayatri ……………

R.No: 101 R.No: 102 R.No:103 …………

C.N: SNIST C.N: SNIST C.N: SNIST ………….

java Test <-enter

1. Start JVM
2. Create and start main thread
3. Locate test.class file
4. Load test.class
5. Execute main() method
6. Unload test.class
7. Terminate main method
8. Shutdown JVM

Static variables will be stored in method area.

We can access static variables either by object reference or by class name but recommended to use class name.

Within the same class it is not required to use class name and we can access directly.

S.opln(Test.x);

S.opln(x);

}

}

We can access static variables directly from both instance and static areas.

class Test

{

Satic int x =10;

Psvm(s args[])

{

S.opln(t.x);

}

Public void m1()

{

S.opln(x);

}

}

For static variables jvm will provide default values and we are not required to perform initialization explicitly.

Class Test

{

Static int x;

Static double d;

Static String s;

Psvm(S args[])

{

S.opln(x); ---------0

S.opln(d); ----------0.0

S.opln(s); ----------null

}

Static variables are also known as class level variables or fields.