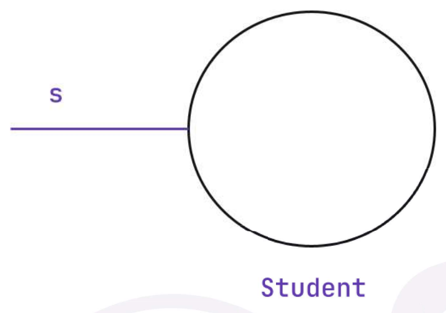
Types of Variables

Division 1 :
Based on the type of value represented by a variable all variables are divided into 2 types. They are:
1. Primitive variables
2. Reference variables

Primitive variables:
Primitive variables can be used to represent primitive values.
Example: int x=10;

Reference variables: Reference variables can be used to refer objects.
Example: Student s=new Student();

Diagram:



Division 2 :
Based on the behavior and position of declaration all variables are divided into the following 3 types.
1. Instance variables
2. Static variables
3. Local variables

Instance variables:
- If the value of a variable is varied from object to object such types of variables are called instance variables.
- For every object a separate copy of instance variables will be created.
- 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 the same as scope of objects.
- Instance variables will be stored on the heap as the part of the object.
- Instance variables should be declared within the class directly but outside of any method or block or constructor.
- Instance variables can be accessed directly from the Instance area. But cannot be accessed directly from a static area.
- But by using object reference we can access instance variables from static area

Example
```
class Test
{
	int i =10;
	public static void main(String[] args) {
		System.out.println(i);//CE: non static variable can’t be referenced

		Test t = new Test();
		System.out.println(t.i);//valid
		t.m1();
	}
	public void m1()
	{
		System.out.println(i);//valid
	}
}
```

Local variables:

- Sometimes to meet temporary requirements of the programmer we can declare variables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.
- Local variables will be stored inside the stack.
- The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly the same as the scope of the block in which we declared.

Example
```
class Test
{
	public static void main(String[] args) {
		int i =0;
		for (int j =0;j≤3 ;j++ )
		{
			i = i+j;
		}
		System.out.println(j);//CE
	}
}
```

- The local variables will be stored on the stack.
- For the local variables JVM won't provide any default values, we should perform initialization explicitly before using that variable.

Example
```
class Test
{
      public static void main(String[] args) {
             int x;
             System.out.println("hello");//hello
     }
}
```

Example
```
class Test
{
      public static void main(String[] args) {
             int x;
             System.out.println(x);//CE
     }
}
```

Example
```
class Test
{
      public static void main(String[] args) {
             int x;
             if(args.length>0)
             {
                    x=10;
             }
             System.out.println(x);//CE
      }
}
```

Example
```
class Test
{
      public static void main(String[] args) {
             int x;
             if(args.length>0)
             {
                    x=10;
             }else{
                    x = 20;
             }
             System.out.println(x);
      }
}
```

- It is never recommended to perform initialization for the local variables inside logical blocks because there is no guarantee of executing that block always at runtime.
- It is highly recommended to perform initialization for the local variables at the time of declaration at least with default values.
- The only applicable modifier for local variables is final. If we are using any other modifier we will get a compile time error.