static
 a. class
 b. variable
 c. method
 d. block

Program to demonstrate the usage of static block , constructor.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
import java.util.Scanner;
class Farmer
{
      //instance variable
      float p,si,t;

      //static variable
      static float r;
      static
      {
            r = 2.5f;
      }

      //instance method
      public void input()
      {
            Scanner scanner = new Scanner(System.in);

            System.out.print("Enter the value of prinicpal amount :: ");
            p = scanner.nextFloat();

            System.out.print("Enter the value of time period:: ");
            t = scanner.nextFloat();
      }

      //instance method
      public void calculateSI()
      {
            si = (p*t*r)/100;
      }

      //instance method
      public void dispSI()
      {
            System.out.println("SI IS :: "+si);
      }
}
class Test
{
      public static void main(String[] args)
      {
            Farmer f1 = new Farmer();
            f1.input();
            f1.calculateSI();
            f1.dispSI();

            System.out.println();

            Farmer f2 = new Farmer();
            f2.input();
```

```
            f2.calculateSI();
            f2.dispSI();

            System.out.println();

            Farmer f3 = new Farmer();
            f3.input();
            f3.calculateSI();
            f3.dispSI();
      }

}

Output
Enter the value of prinicpal amount :: 30000
Enter the value of time period:: 5
SI IS :: 3750.0

Enter the value of prinicpal amount :: 40000
Enter the value of time period:: 6
SI IS :: 6000.0

Enter the value of prinicpal amount :: 50000
Enter the value of time period:: 8
SI IS :: 10000.0


Write a program to count the no of objects created for a class?
class Test
{
      //static variable
      static int count;

      Test(){}
      Test(int i){}
      Test(int i, int j){}
      Test(int i,int j, int k){}

      //instance block
      {
            count++;
      }

      //instance method
      public void disp()
      {
            System.out.println("No of objects created is :: "+count);
      }
      public static void main(String[] args)
      {
            Test t1 = new Test();
            Test t2 = new Test(10);
            Test t3 = new Test();
            Test t4 = new Test(100);
            Test t5 = new Test(100,200);
            Test t6 = new Test(100,200,300);

            t1.disp();
      }
```

```
}

Output::
No of objects created is :: 6

Using var-args
++++++++++++++
class Test
{
      //static variable
      static int count;

      Test(int... args){
            count++;
      }

      //instance method
      public void disp()
      {
            System.out.println("No of objects created is :: "+count);
      }
      public static void main(String[] args)
      {
            Test t1 = new Test();
            Test t2 = new Test(10);
            Test t3 = new Test();
            Test t4 = new Test(100);
            Test t5 = new Test(100,200);
            Test t6 = new Test(100,200,300);

            t1.disp();
      }
}
Output::
No of objects created is :: 6



final variables
 a. final instance variable
 b. final static variable
 c. final local varaible

Note: A variable is said to be final iff the value of the variable is "fixed", we
can't change the value once it is initialized.
      If a variable is final, then compiler will get to know the value of the
variable and these values will be used during the evaluation
      of an Expression.


eg#1.
class Test
{
      int i;
      public static void main(String[] args)
      {
            System.out.println(new Test().i);
      }
}
```

Output
 0

If an instance variable is marked as final, then for such instance variables, jvm will not give default value, programmer should supply the
value for instance variables, otherwise it would result in "CompileTimeError".
eg#2.

```
class Test
{
      final int i;
      public static void main(String[] args)
      {
            System.out.println(new Test().i);
      }
}
```

Output
 CompileTimeError:: variable i not initialized in the default constructor

Places to initalize the value for final instance variable
++
1. At the time of declaration

eg#1

```
class Test
{
      final int i=100;
      public static void main(String[] args)
      {
            System.out.println(new Test().i);//100
      }
}
```

2. Inside instance block
eg#1

```
class Test
{
      final int i;
      {
            i = 100;
      }
      public static void main(String[] args)
      {
            System.out.println(new Test().i);//100
      }
}
```

3. Inside constructor
eg#1

```
class Test
{
      final int i;

      Test()
      {
            i = 100;
      }
      public static void main(String[] args)
      {
```

```
            System.out.println(new Test().i);//100
      }
}
```
Apart from these 3 places, if we try to do initalization then it would result in "CompiletimeError".

```
eg#1.
class Test
{
      final int i;

      public void m1()
      {
            i = 100;//CE: error: cannot assign a value to final variable i
      }
      public static void main(String[] args)
      {
            System.out.println(new Test());

      }
}
```

static variable
+++++++++++++++

```
eg#1.
class Test
{
      static int i;
      public static void main(String[] args)
      {
            System.out.println(i);//0
      }
}
```

If an static variable is marked as final, then for such static variables, jvm will not give default value, programmer should supply the
value for static variables, otherwise it would result in "CompileTimeError".

```
eg#2.
class Test
{
      final static int i;
      public static void main(String[] args)
      {
            System.out.println(i);
      }
}
```

Output : CE: variable i not initialized in the default constructor

Places to initialize final static variable
++
1. At the time of declaration

```
eg#1.
class Test
{
      final static int i=100;
      public static void main(String[] args)
      {
```

```
            System.out.println(i);//100
      }
}
```

2. Inside static block

```
eg#2.
class Test
{
      final static int i;
      static
      {
            i = 100;
      }
      public static void main(String[] args)
      {
            System.out.println(i);//100
      }
}
```

If a variable is marked as static and final, then for those variables initialization should be completed before class loading completes,otherwise it would result in "CompileTimeError".

```
eg#3.
class Test
{
      final static int i;

      Test()
      {
            i = 100;//CE: cannot assign a value to final variable i
      }
      public static void main(String[] args)
      {
            System.out.println(i);
      }
}
```

final local variable

++++++++++++++++++++

```
eg#1.
class Test
{
      public static void main(String[] args)
      {
            int i;
            System.out.println(i);
      }
}
```

Output: error: variable i might not have been initialized

```
eg#2.
class Test
{
      public static void main(String[] args)
      {
            final int i;
            System.out.println(i);
```

```
	}
}
```
Output: error: variable i might not have been initialized

The only place where we can initialize the local variable is at the time of declaration.

eg#1.
```
class Test
{
	public static void main(String[] args)
	{
		final int i=100;
		System.out.println(i);//100
	}
}
```

eg#2.
```
class Test
{
	final static int i;
	int j;

	static
	{
		i =10;
	}

	{
		j = 100;
	}
	Test()
	{
		j = 200;
	}
	public static void main(String[] args)
	{
		final int k = 300;
		System.out.println(i);
		System.out.println(new Test().j);
		System.out.println(k);
	}
}
```
Output
```
10
200
300
```

eg#3.
```
class Test
{
	public static void main(String[] args)
	{
		methodOne(100,200);
	}
	public static void methodOne(final int i, int j)
	{
		i = 1000;
```

```
            j = 2000;
            System.out.println(i + " " + j);
      }
}
Output :: error: final parameter i may not be assigned
                i = 1000;
                ^
```

Note: The only access modifier applicable at local variable level is "final".

Inheritance in java
++++++++++++++++++
=> It refers to process of linking 2 classes.
=> In java inheritance can be promoted in 2 ways
      a. IS-A relationship
      b. HAS-A relationship
=> IS-A relationship is referred to "Inheritance".

What is inheritance?
 It refers to process of acquiring properties and behaviours from parent to child class.

What is the benefit of inheritance?
 => Re-Usablitiy.

IS-A relationship to promote in java we use "extends" keyword.

eg#1.

```
class Parent
{
      public void methodOne()
      {
            //body of the method
      }
}

class Child extends Parent
{
      //inherited method
      public void methodOne()
      {
            //body of the method
      }

      //Specialized method
      public double methodTwo()
      {
            //body of the method
            return 20.0;
      }

}
```

eg#2.

```
class Parent
{
```

```
	public void methodOne()
	{
		System.out.println("Parent method");
	}
}
class Child extends Parent
{
	//inherited method(public category from parent)
	public void methodOne()
	{
		System.out.println("Parent method");
	}

	//Specialized method
	public void methodTwo()
	{
		System.out.println("Child method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		Parent p= new Parent();
		p.methodOne();
		p.methodTwo();//CE: can't find symbol

		System.out.println();

		Child c = new Child();
		c.methodOne();
		c.methodTwo();

		System.out.println();

		Parent  p1 =new Child();
		p1.methodOne();
		p1.methodTwo();//CE: can't find symbol

		System.out.println();

		Child c1 = new Parent();//incompatible types

	}
}

Output
D:\OctBatchMicroservices\Test.java:22: error: cannot find symbol
				p.methodTwo();//CE: can't find symbol
				 ^
  symbol:   method methodTwo()
  location: variable p of type Parent

D:\OctBatchMicroservices\Test.java:34: error: cannot find symbol
				p1.methodTwo();//CE: can't find symbol
				  ^
  symbol:   method methodTwo()
  location: variable p1 of type Parent
```

```
D:\OctBatchMicroservices\Test.java:38: error: incompatible types: Parent cannot be
converted to Child
                Child c1 = new Parent();//incompatible types
```

Note:
Whatever the parent class has under public category, by default will be available to child class.
Whatever the child class has under public categroy, by default won't be available to the parent class.
Using the parent reference,we can make a call only to the parent class methods but not the child class specialized methods.
Parent class reference can be used to collect child class objects,but by using parent class reference we can call only parent class methods
but not child class specialized methods.
Child class reference can't be used to hold parent class objects.

Realtime usecases of inheritance[Code Re-Usability]

```
++++++++++++++++++++++++++++++++++++++++++++++++++
class Loan
{
      //common methods available for any type of Loan
}
class HousingLoan extends Loan
{
      //Housing loan specifc methods
}
class VehicleLoan extends Loan
{
      //Vehicle loan specifc methods
}
class EducationLoan extends Loan
{
      //Education loan specifc methods
}
```

eg#2.

```
class Object
{
  public final native java.lang.Class<?> getClass();
  public native int hashCode();
  protected native java.lang.Object clone() throws
java.lang.CloneNotSupportedException;

  //used while working with String,StringBuilder,StringBuffer
  public boolean equals(java.lang.Object);

  //toString() gets called automatically when we print the reference of the
object[callback method/magic method]
  public java.lang.String toString();

  //Methods related to MultiThreading
  public final native void notify();
  public final native void notifyAll();
  public final void wait() throws java.lang.InterruptedException;
  public final native void wait(long) throws java.lang.InterruptedException;
```

```
  public final void wait(long, int) throws java.lang.InterruptedException;

  //Method related to GarbageCollector
  protected void finalize() throws java.lang.Throwable;

}

class String extends Object
{
      //String class specific methods
      @Override
      public String toString()
      {
            //print the content of the Strings
      }
}

class StringBuffer extends Object
{
      //StringBuffer class specific methods
      @Override
      public String toString()
      {
            //print the content of the Strings
      }
}

class StringBuilder extends Object
{
      //StringBuilder class specific methods

      @Override
      public String toString()
      {
            //print the content of the Strings
      }
}
```

Note:
1. For all java classes,, the most commonly requried functionality is defined inside object class so Object class is called as "root" for all java classes.
2. For all java exceptions and errors, the most commonly required functionality is defined inside "Throwable" class, so Throwable class acts as root for "Exception Hierarchy".
3. For compiler and jvm by default all the classes available in a package/folder called "java.lang.*" will available.

eg#1.

```
class Student
{
      int id;
      String name;

      Student(int id,String name)
      {
            this.id = id;
            this.name = name;
```

```
	}

	/*
	public java.lang.String toString()
	{
		print the reference/hashcode of the object
	}
	*/
}

class Test
{
	public static void main(String[] args)
	{
		Student std = new Student(10,"sachin");
		System.out.println(std);//std.toString():: hashCode

		System.out.println();

		String str = new String("sachin");
		System.out.println(str);//str.toString():: sachin
	}
}
```

Snippets
++++++++
Given:

```
1. public class Barn {
2.    public static void main(String[] args) {
3.          new Barn().go("hi", 1);
4.          new Barn().go("hi", "world", 2);
5.    }
6.    public void go(String... y, int x) {
7.          System.out.print(y[y.length - 1] + " ");
8.    }
9. }
```

What is the result?
A. hi hi
B. hi world
C. world world
D. Compilation fails.
E. An exception is thrown at runtime.

Answer: D

Q>

```
public class Yippee {
	public static void main(String [] args) {
		for(int x = 1; x < args.length; x++) {
			System.out.print(args[x] + " ");
		}
	}
}
```

and two separate command line invocations:

```
java Yippee
java Yippee 1 2 3 4
```

What is the result?

A. No output is produced.
   1 2 3
B. No output is produced.
   2 3 4
C. No output is produced.
   1 2 3 4
D. An exception is thrown at runtime.
   1 2 3
E. An exception is thrown at runtime.
   2 3 4
F. An exception is thrown at runtime.
   1 2 3 4

Anwer: No Output is produced
       2 3 4

Q>
```
public static void test(String str) {
      int check = 4;
      if (check = str.length()) {
            System.out.print(str.charAt(check -= 1) +", ");
      } else {
            System.out.print(str.charAt(0) + ", ");
      }
}
```
and the invocation:
```
test("four");
test("tee");
test("to");
```
What is the result?

A. r, t, t,
B. r, e, o,
C. Compilation fails.
D. An exception is thrown at runtime.

Answer: C