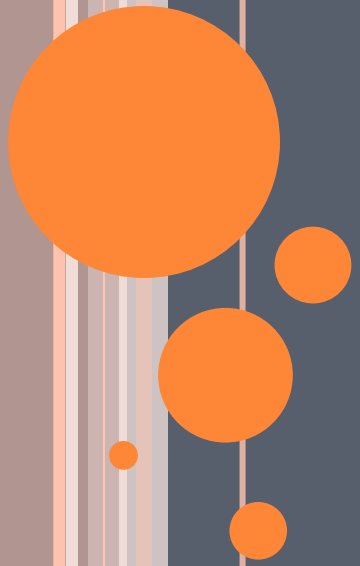
JAVA

Agenda

- static
- Final
- enum



Static

STATIC VARIABLE

- The static variable can be used to refer the common property of all objects (that is not unique for each object)
  - e.g. company name of employees,college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.
- It makes the program **memory efficient** (i.e it saves memory).

```
class Student{
    int rollno;
    String name;
    String college=“BITS";
}
```

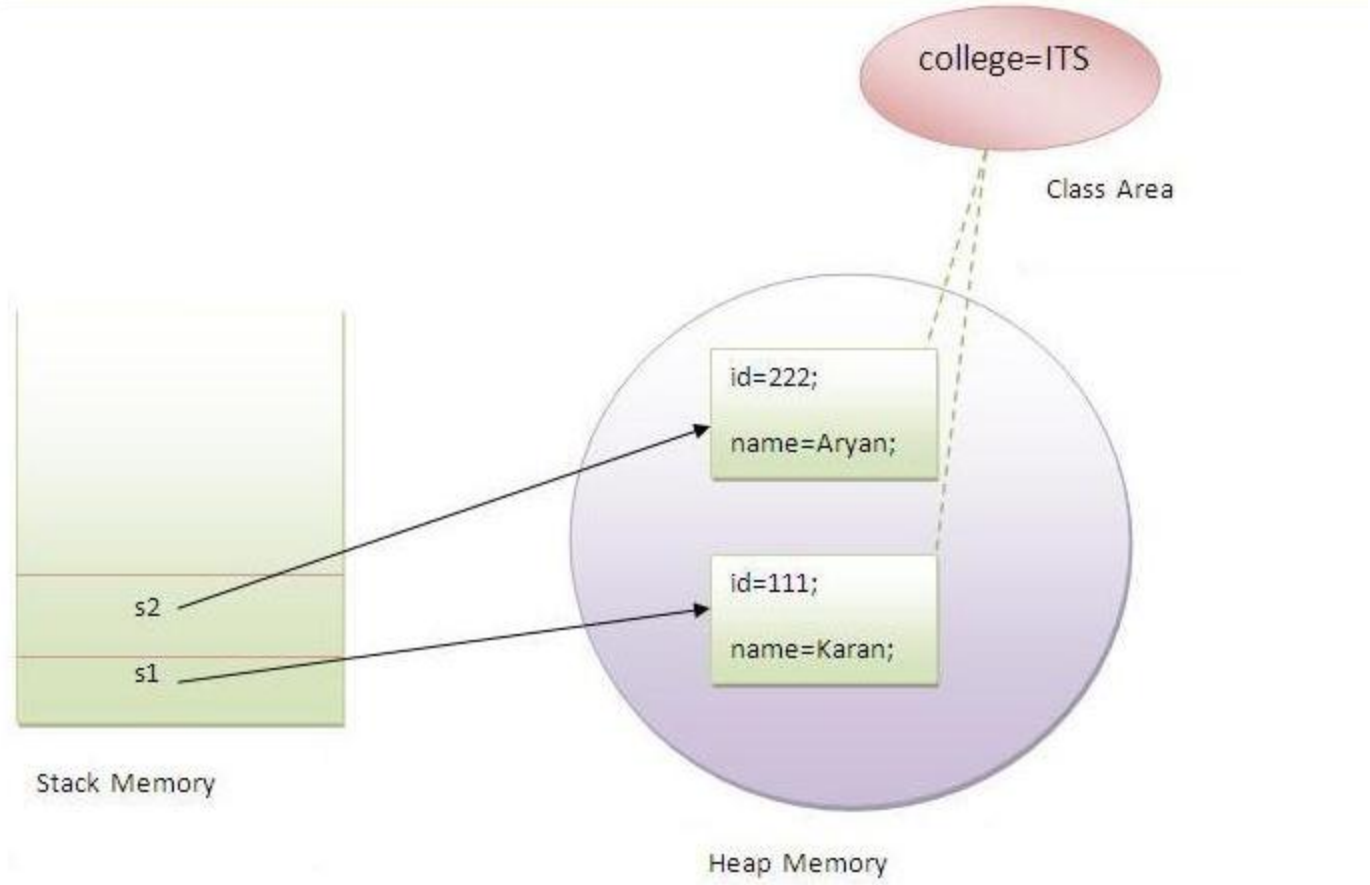
```
//Program of static variable

class Student{
   int rollno;
   String name;
   static String college =“BITS";

   Student(int r,String n){
   rollno = r;
   name = n;
   }
 void display (){System.out.println(rollno+" "+name+" "+college);}

 public static void main(String args[]){
 Student s1 = new Student(111,“Aryan");
 Student s2 = new Student(222,”Karan");

 s1.display();
 s2.display();
 }
}
```


college=ITS
Class Area
id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

STATIC METHOD

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.

```
//Program to get cube of a given number by static method

class Calculate{
  static int cube(int x){
  return x*x*x;
  }

  public static void main(String args[]){
  int result=Calculate.cube(5);
  System.out.println(result);
  }
}
```

```
class Student
{
    int rollno;
    String name;
    static String college = “BITS";

    static void change(){
    college = “PSG";
    }

    Student(int r, String n){
    rollno = r;
    name = n;
    }

    void display (){System.out.println(rollno+" "+name+" "+college);}

   public static void main(String args[]){
    Student.change();

    Student s1 = new Student (111,“ABC");
    Student s2 = new Student (222,“XYZ");
    Student s3 = new Student (333,“LMN");

    s1.display();
    s2.display();
    s3.display();
    }
}
```

Restrictions for static method

- The static method can not use non static data member or call non-static method directly.
- this and super cannot be used in static context.

STATIC BLOCK

- Is used to initialize the static data member.
- It is executed before main method at the time of classloading.
- Example

```
class A2{

  static{System.out.println("static block is invoked");}

  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```

Final

Final

- Used to restrict the usage
- The final keyword can be used with
  - variable
  - method
  - class

Final Variable

Constants are static final variables.

```
public class Bank {
  private static final double  DEFAULT_INTEREST_RATE = 3.2;
  ... // more declarations
}
```

Blank Final Variable

```
public class Customer {

  private final long customerID;

  public Customer() {
    customerID = createID();
  }

  public long getID() {
    return customerID;
  }

  private long createID() {
    return ... // generate new ID
  }

  // more declarations

}
```

Final Class

```
final class Bike{}

 class Honda1 extends Bike
{
void run()
{
System.out.println("running safely with 100kmph");
}

   public static void main(String args[]){
   Honda1 honda= new Honda();
   honda.run();
   }
 }
```

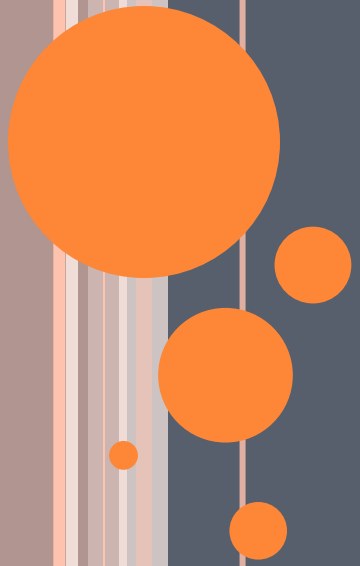
Final Method

- A method can be declared final if it has an implementation that should not be changed and it is critical to the consistent state of the object.
- A class that is declared final cannot be sub-
- If you make any method as final, you cannot override it. classed.
- During Inheritance , final method is inherited but you cannot override it

Final Parameter

```
class Bike11{
  int cube(final int n){
   n=n+2;//can't be changed as n is final
   n*n*n;
  }
  public static void main(String args[]){
    Bike11 b=new Bike11();
    b.cube(5);
 }
}
```

- You cannot subclass a `final` class.
- You cannot override a `final` method.
- A `final` variable is a constant.
- You can set a `final` variable once only, but that assignment can occur independently of the declaration; this is called a *blank final variable*.
  - A blank final instance attribute must be set in every constructor.
  - A blank final method variable must be set in the method body before being used.



Enum

Enum

- An enum is a data type which contains fixed set of constants.
  - It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY)
  - directions (NORTH, SOUTH, EAST and WEST) etc.
- Are static and final implicitly.
- Available from Java 5.
- Enums can be thought of as classes that have fixed set of constants.

Features

- enum improves type safety
- enum can be easily used in switch
- enum can be traversed
- enum can have fields, constructors and methods
- enum may implement many interfaces but cannot extend any class because it internally extends Enum class

Enum Sample

```
class EnumExample1
{
public enum Season
   { WINTER, SPRING, SUMMER, FALL }

public static void main(String[] args)
{
for (Season s : Season.values())
System.out.println(s);
}

}
```

VALUES() METHOD

- The java compiler internally adds the values() method when it creates an enum.
- The values() method returns an array containing all the values of the enum.

```
public static final class EnumExample1$Season extends Enum
{
  private EnumExample1$Season(String s, int i)
  {
    super(s, i);
  }

  public static EnumExample1$Season[] values()
  {
    return (EnumExample1$Season[])$VALUES.clone();
  }

  public static EnumExample1$Season valueOf(String s)
  {
    return (EnumExample1$Season)Enum.valueOf(EnumExample1$Season, s);
  }
```

```
  public static final EnumExample1$Season WINTER;

  public static final EnumExample1$Season SPRING;
  public static final EnumExample1$Season SUMMER;

  public static final EnumExample1$Season FALL;
  private static final EnumExample1$Season $VALUES[];

  static
  {
    WINTER = new EnumExample1$Season("WINTER", 0);
    SPRING = new EnumExample1$Season("SPRING", 1);
    SUMMER = new EnumExample1$Season("SUMMER", 2);
    FALL = new EnumExample1$Season("FALL", 3);
    $VALUES = (new EnumExample1$Season[] {
      WINTER, SPRING, SUMMER, FALL
    });
  }

}
```

Initializing specific value to the enum constants

- The enum constants have initial value that starts from 0, 1, 2, 3 and so on.
- But we can initialize the specific value to the enum constants by defining fields and constructors.
- Enum can have fields, constructors and methods.

Initializing specific value to the enum constants

```
class EnumExample4
{
enum Season
{
WINTER(5), SPRING(10), SUMMER(15), FALL(20);

private int value;
private Season(int value)
{this.value=value;}

}
public static void main(String args[])
{
for (Season s : Season.values())
System.out.println(s+" "+s.value);
}
}
```

Applying enum on switch statement

```
class EnumExample5
{
enum Day{ SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY}

public static void main(String args[]){

Day day=Day.MONDAY;

switch(day){
case SUNDAY:
 System.out.println("sunday");
 break;
case MONDAY:
 System.out.println("monday");
 break;
default:
System.out.println("other day");
}

}}
```