Static, Final, String, Object

Session Objective

- Static
- Final
- String & String Buffer
- Object class

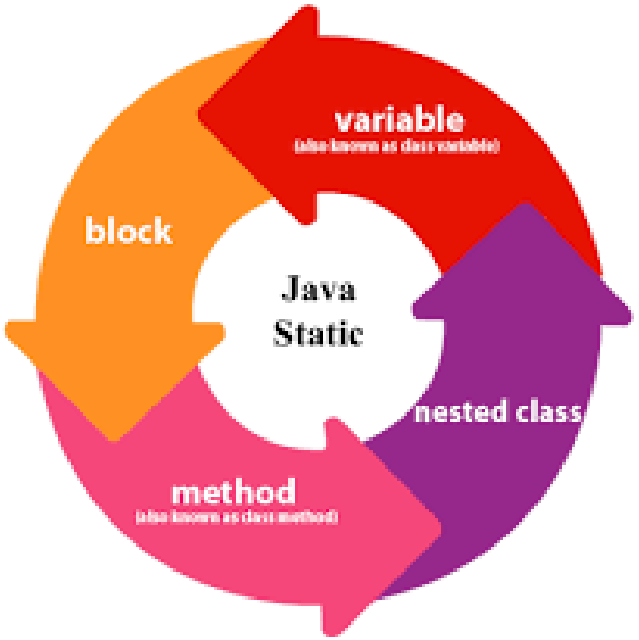
Static & Final

Static Contd…

Static keyword in Java is used for memory management

The static can be:

- Variable (also known as a class variable)
- Method (also known as a class method)
- Block
- Nested class



Static Contd…

Java static variable

- The static variable can be used to refer to the common property of all objects
- The static variable gets memory only once in the class area at the time of class loading
- Static variables are shared among all the instances of class

```
String static int empid=31410;
```


Static Contd…

Advantages of static variable

It makes the program **memory efficient**

Example:

There are 1000 employees in a MARK company, now all instance data members will get memory each time when the object is created. All the employees will have their unique employee id and name, so instance data member is good in such case. Where as, “company name" refers to the common property of all objects. If it is made static, then the field will get the memory only once.

Static Contd…

Points to Remember:

- Static variables belong to a class, they can be accessed directly using class name and don't need any object reference
- Static variables can only be declared at the class level
- Static fields can be accessed without object initialization

Static Contd…

The static Methods

Static methods also belong to a class instead of the object, and so they can be called without creating the object of the class in which they reside.

- ✓Static methods in Java are resolved at compile time. Since method overriding is part of Runtime Polymorphism, so static methods can't be overridden
- ✓Abstract methods can't be static
- ✓Static methods cannot use **this** or **super** keywords

```
public static void fun(){
        System.out.println("Employee id:"+empid);
}
```


Static Contd…

A static Block

- A static block is used for initializing static variables
- This block gets executed when the class is loaded in the memory.
- A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.

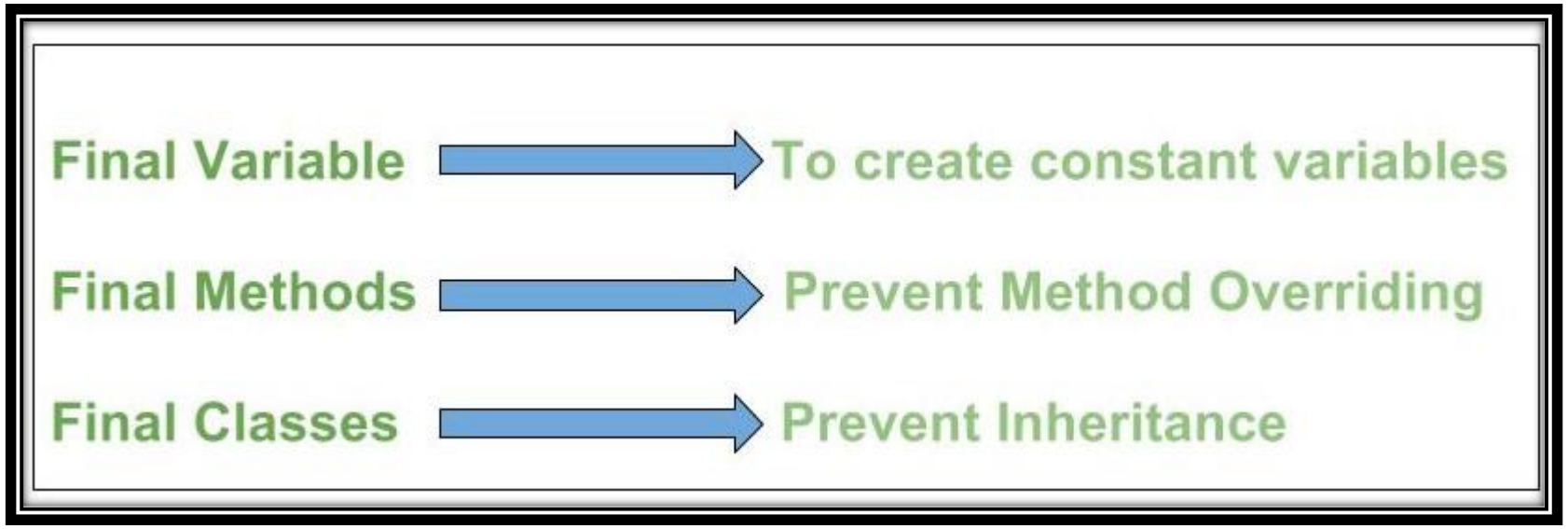
```
static {
System.out.println("Static block");
}
```


Static

Contd…

```
class Exam{
    static String tech="Java";
}
class StaticDemo{
    static {
        System.out.println("I am static");
    }
    static String int empid=31410;
    String int age=20;
public   void fun(){
        System.out.println("Employee id:"+empid);
System.out.println("Age:"+ ++age);
    }
    public static void main(String[] args) {
        System.out.println(empid);
        System.out.println(Exam.tech);
        StaticDemo demo=new StaticDemo();
        demo.fun();
        fun();
    }
}
```

Final in java



Final

```
class FinalDemo{
    final int empid=31410;
    final int age=20;
}
```


String & String Buffer

String, String Buffer, String Builder

- String is immutable in Java, so it’s easy to share it across different threads or functions.
- whenever String manipulation like concatenation, substring etc is done, it generates a new String and discards the older String for garbage collection.
- String class represents character strings

Note:

- A String represents, a string in the UTF-16 format
- String is a final class

String Contd…

Instantiate String by two ways.

1. String str = "abc";
2. String str = new String ("abc");

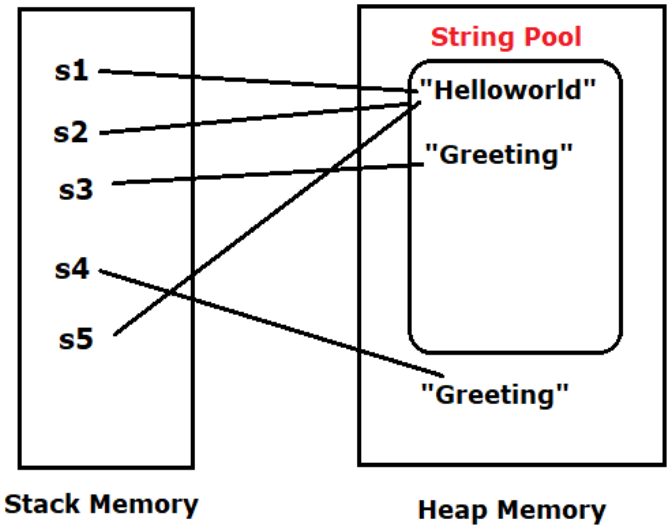
- String class overrides equals() and hashcode() method
- toString() method provides String representation of any object
- It is declared in Object class

String Contd…

Problem with String in Java

- String is used to perform lots of operation on them e.g. converting a string into uppercase, lowercase, getting substring out of it , concatenating with other string etc. Since String is an immutable class every time a new String is created and the older one is discarded which creates lots of temporary garbage in the heap.

```
ing s1 = "Helloworld";
ing s2 = "Helloworld";
ing s3 = "Greeting";
ing s4 = new String("Greeting");
ing s5 = "Helloworld";
```



String Contd…

- StringBuffer and StringBuilder are mutable objects in java and provide append(), insert(), delete() and substring() methods for String manipulation.
- Java String class provides a lot of methods to perform operations on string such as
  - compare()
  - concat()
  - equals()
  - split()
  - length()
  - replace()
  - compareTo()
  - intern()
  - substring()

String vs StringBuffer vs StringBuilder

	String	StringBuffer	StringBuilder
Storage	String pool	Heap	Heap
Modifiable	No(immutable)	Yes (mutable)	Yes (mutable)
Thread safe	Yes	Yes	No
Synchronized	Yes	Yes	No
Performance	Fast	Slow	Fast

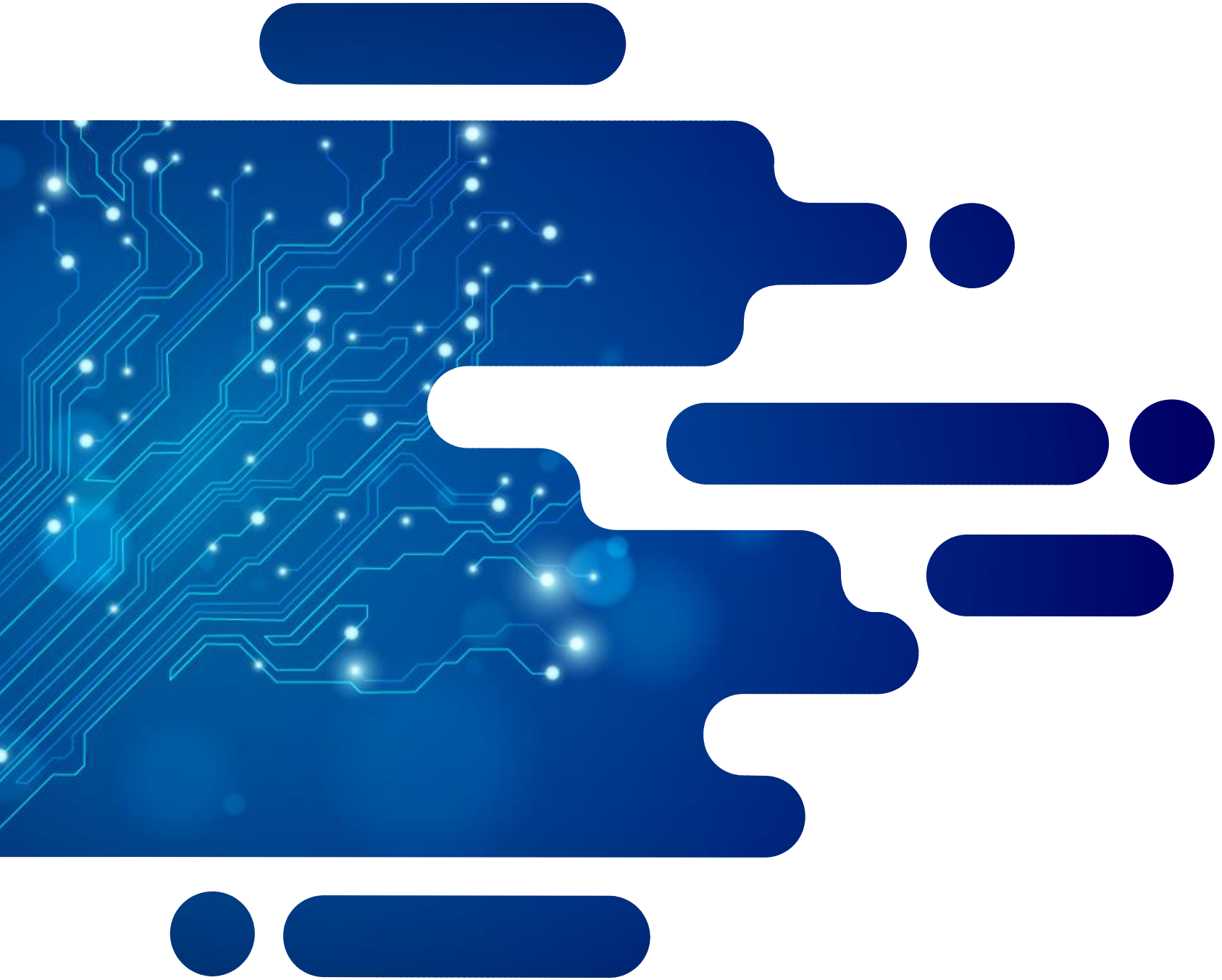
String

Contd…

```
public class Email_Validation {
    public static void main(String[] args) {
        String email = "admin@hexaware.com";
        boolean checkEndDot  = false;
        checkEndDot = email.endsWith(".");
        int indexOfAt = email.indexOf('@');
        int lastIndexOfAt = email.lastIndexOf('.');
        int countOfAt = 0;

for (int i = 0; i < email.length(); i++) {
            if(email.charAt(i)=='@')
                countOfAt++; }
        String buffering = email.substring(email.indexO
f('@')+1, email.length());
        int len = buffering.length();
        int countOfDotAfterAt = 0;
```

```
            for (int i=0; i < len; i++) {
           if(buffering.charAt(i)=='.')
              countOfDotAfterAt++; }
           String userName = email.substring(0, email.indexOf('@'));
           String domainName = email.substring(email.indexOf('@')
+1, email.length());
              System.out.println("\n");
             if ((countOfAt==1) && (userName.endsWith(".")==false)
 && (countOfDotAfterAt ==1) &&
                ((indexOfAt+3) <= (lastIndexOfAt) && !checkEndDot))
{
                 System.out.println("\"Valid email address\"");}
             else {
                        System.out.println("\n\"Invalid email address\""
); }
              System.out.println("\n");
              System.out.println("User name: " +userName+ "\n" + "
Domain name: " +domainName);
    }
}
```



Object class

Object Class

- Object class is present in **java.lang** package.
- Every class in Java is directly or indirectly derived from the **Object** class.

Object class methods

- **toString()**
- **equals()**
- **hashcode()**

equals() Method

- equals(Object obj) – It is used to simply verify the equality of two objects.
- It’s default implementation check the object references of two objects to verify their equality.
- By default, two objects are equal if and only if they are stored in the same memory address.

hashCode() Method

- If two Objects are equal, according to the equals(Object) method, then hashCode() method must produce the same Integer on each of the two Objects.
- hashcode() – Returns a unique integer value for the object in runtime. By default, integer value is mostly derived from memory address of the object in heap (but it’s not mandatory always).
- If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.

toString() Method

- **toString()** : toString() provides String representation of an Object and used to convert an object to String.
- Whenever any Object reference is printed, then internally toString() method is called.

```
Product product=new Product();
System.out.println(product);
```


Example for toString(),equals() & hashcode()

```
public class Training{
    private int trgId;
    private String trgName;
    public Training(){}
    public Training(int trgId,String trgName){
        this.trgId=trgId;
        this.trgName=trgName;
    }
//generate getter and setter methods
public String toString(){
        return "Training id:"+trgId+"Traininig Name:"+trgName;
    }
```


Example for toString(),equals() & hashcode()

```
@Override
    public final boolean equals(final Object obj) {
     Training trg=(Training)obj;
      if(Objects.equals(trgId,trg.trgId) && Objects.equals(trgName,trg.trgName)) {
        return true;
      }
      return false;
    }
    @Override
    public final int hashCode() {
      return Objects.hash(trgId, trgName);
    }
}
```


Difference between == and .equals()

- The operator == is used for reference comparison (address comparison).
- The method .equals() is used for content comparison.
- In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.

Example for == & .equals()

```
String msg1=new String("welcome");
  String msg2=new String("welcome all");
   if(msg1==msg2){
      System.out.println("true");
   }else{
      System.out.println("false");
   }
   if(msg1.equals(msg2)){
    System.out.println("true");
   }else{
    System.out.println("false");
  }
```


thank you

www.hexaware.com