Overview

Core Java for Selenium Automation Beginners – II

Methods , Classes and Object Introduction in java for selenium

Create class , Methods parameterized and non parameterized , with and without return type , Create object of class etc

Object oriented programming for selenium

Introduction to Opps and detail understanding of Inheritance,Polimorphism, Encapsulation , Abstraction

Final Keyword in Java

Using final for Class , Method , Variables

String , String Buffer and String Builder for selenium

String Class methods

Exception Handling in Java for Selenium

Checked and Unchecked Exception , Try, Catch , Finally , Throws , Throw etc

Collection Framework in java for selenium

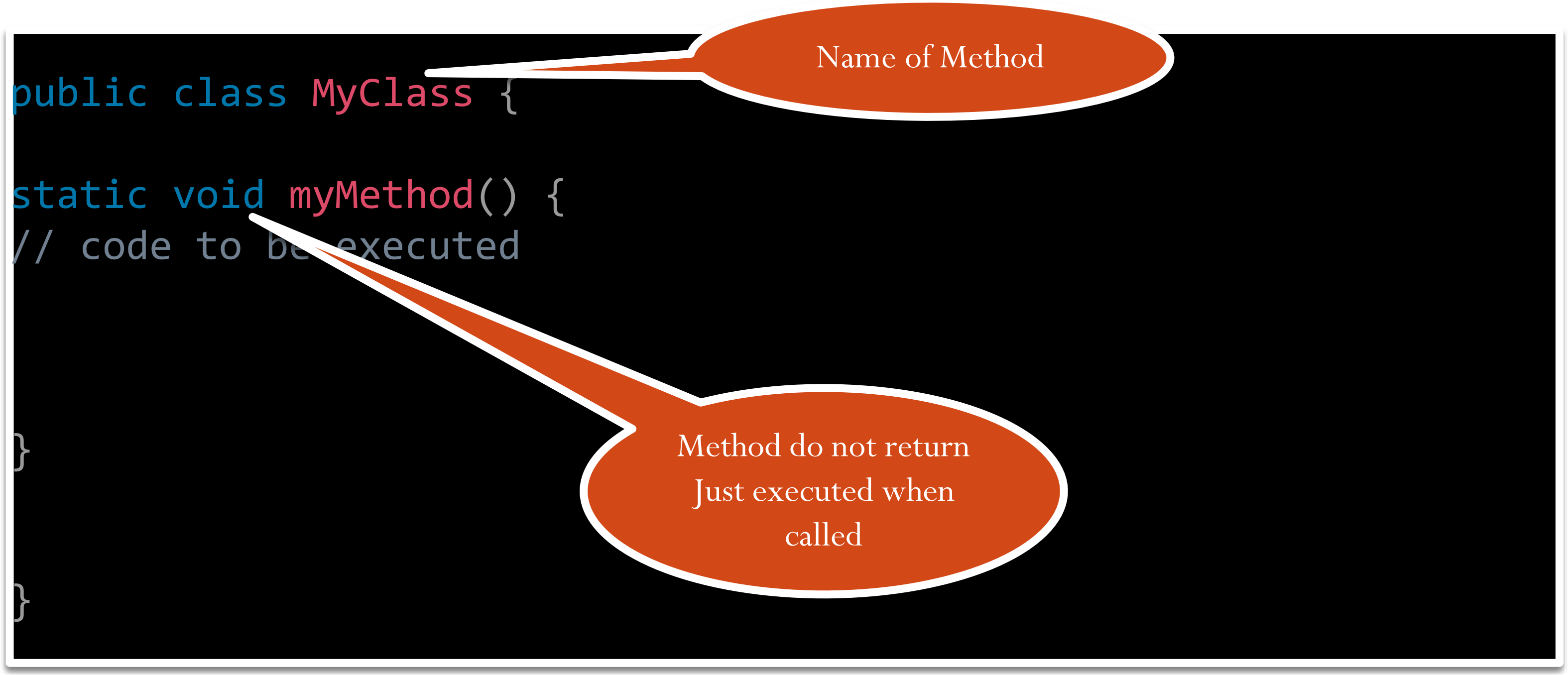
Array List , Hash Set , Hash Map

Methods in Java for Selenium

Java methods

- A **method** is a block of code which only runs when it is called.
- You can pass data, known as **parameters**, into a method.

Create Method inside class



```
public class MyClass {

static void myMethod() {
// code to be executed



}


}
```

Java methods – Call Method

```
public class MyClass {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

  public static void main(String[] args) {

 myMethod();

 myMethod();

 myMethod();
  }
}W
```

You can call method multiple times !

method parameters or arguments

```
public class MyClass {
  static void myMethod(String firstName,String lastName) {
    System.out.println(“Your Full Name is : ” + (firstName+ lastName));
  }

  public static void main(String[] args) {

 myMethod(“Sandip”, “Akolkar”);
 myMethod(“Arohi”, “Akolkar”);

}
}
```

Parameters act as variables inside the method.

Information can be passed to methods as parameter.

method can return values

```
public class MyClass {
  static int  myMethod(int a,int b) {
    int sum=a+b;
    return sum;
}

  public static void main(String[] args) {

System.out.println (myMethod(2, 4));
System.out.println (myMethod(5, 4));

}
}
```

- Use a primitive data type (such as int, char, etc.) instead of ***void***
- Use the **return** keyword inside the method

method demo with if else

Java Classes and Objects

- Java is an **object-oriented** programming language.
- A Class is like an object constructor, or a "blueprint" for creating objects
- Object has **state/properties** and **behavior**
- Everything in Java is associated with classes and objects

E.g. Car – **Object**
weight and color- **data members, properties , attributes OR variables**
drive , refuel – **methods , functions**

.

Introduction to object oriented programming in Java for Selenium

Create object of class and access data members

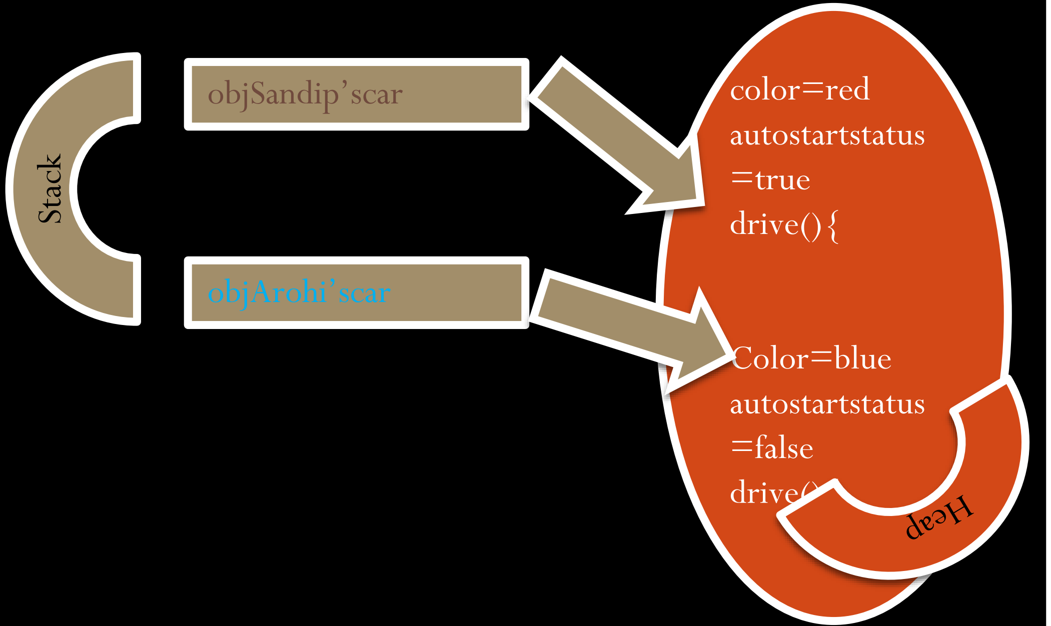
```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {

MyClass myObj = new MyClass();

                    System.out.println(myObj.x);
  }
}
```

ClassName ObjName= new ClassName()

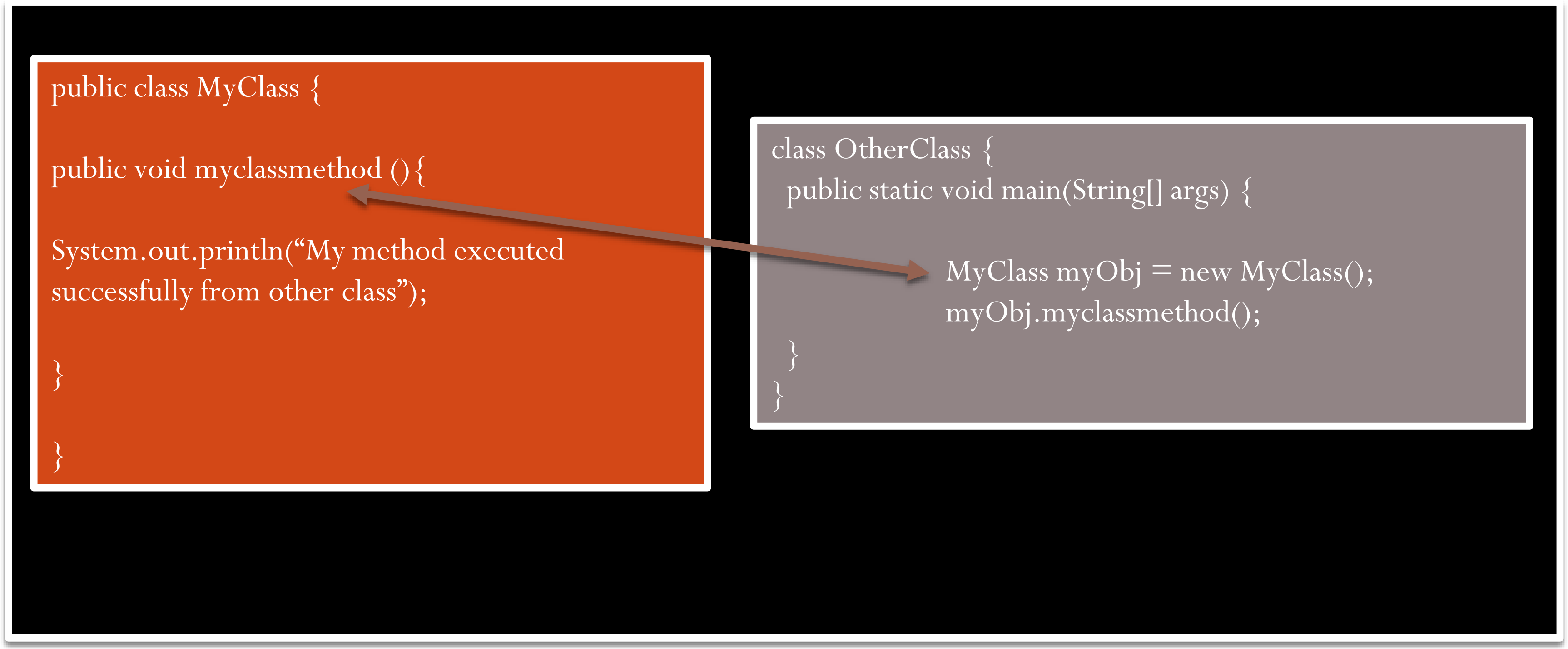
- Objname is variable name or reference
- new + className() is Object together

Object and memory allocation

```
public class MyCar{
                    String Color;
                    boolean autostartstatus;
                    static void  drive(){
System.out.println("I am driving car  having"  + color );
                    }

  public static void main(String[] args) {

MyCar objSandip'scar = new MyCar();
MyCar objArohi'scar = new MyCar();

objSandip'scar.color="red";
objSandip'scar.autostartstatus=true;
bjSandip'scar..drive();

objArohi'scar.color='blue';
objArohi'scar.autostartstatus=false;
objArohi'scar.drive();
  }
}
```



Other class object and my class methods



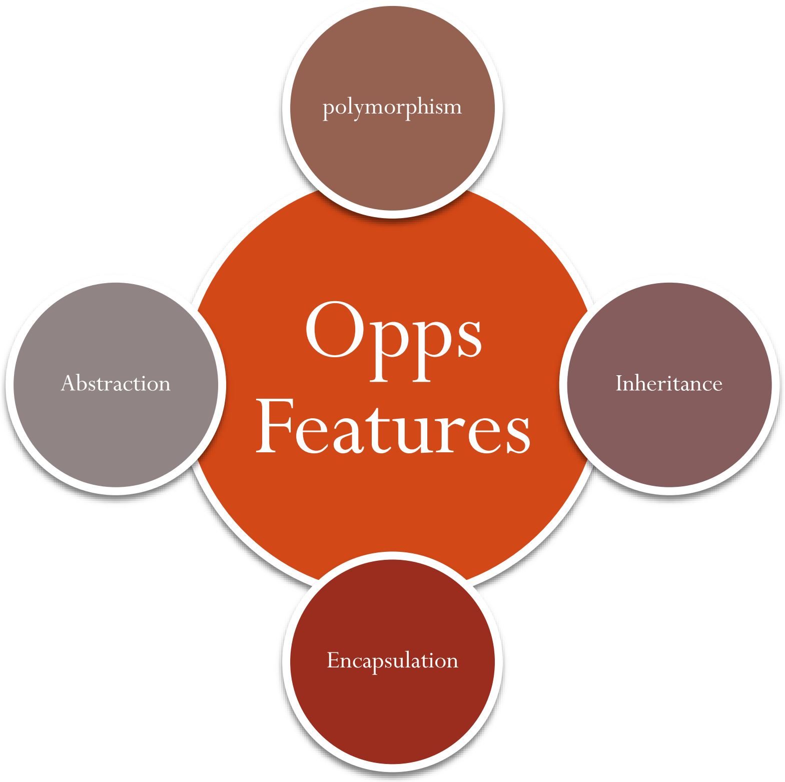
```
public class MyClass {

public void myclassmethod (){

System.out.println("My method executed
successfully from other class");

}

}
```

```
class OtherClass {
  public static void main(String[] args) {

            MyClass myObj = new MyClass();
            myObj.myclassmethod();
  }
}
```

Why Object Oriented Programming

- Objective of Object Oriented programming is to bind data and functions together
- Majorly functions will access the data .

Object Oriented Programming features



Inheritance in Java for Selenium

Inheritance in java for selenium

- Inherit variables and methods from one class to another
  - **subclass** (child) - the class that inherits from another class
  - **superclass** (parent) - the class being inherited from

- use the extends keyword to inherit

- Types – Single , Multilevel , Hierarchical

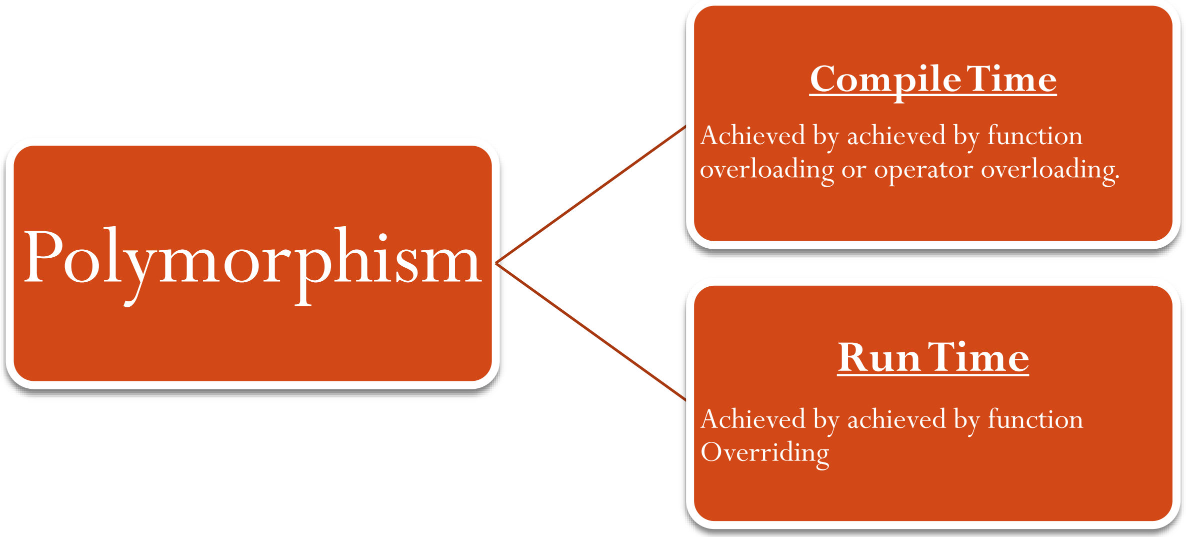
- Single – Class B Extends Class A
- Multilevel - Class C extends class B and class B extends class A
- Hierarchical - B, C & D extends the same class A.

Demo

Polymorphism in Java for Selenium

Polymorphism in Java for Selenium

- Polymorphism means on thing having many forms
- **Real life example of polymorphism-** Man at the same time is a father, a husband, an employee
- The word “poly” means many and “morphs” means forms



Polymorphism

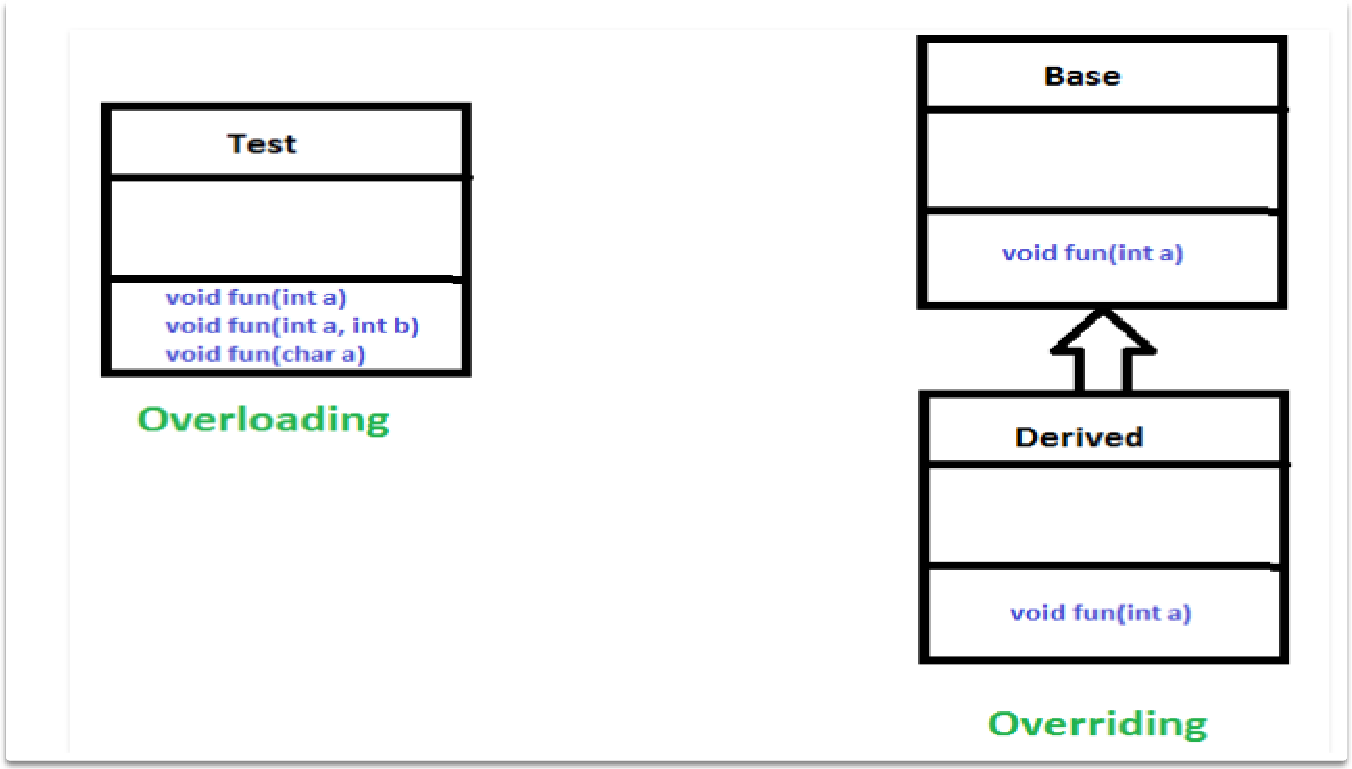
Compile Time

Achieved by achieved by function overloading or operator overloading.

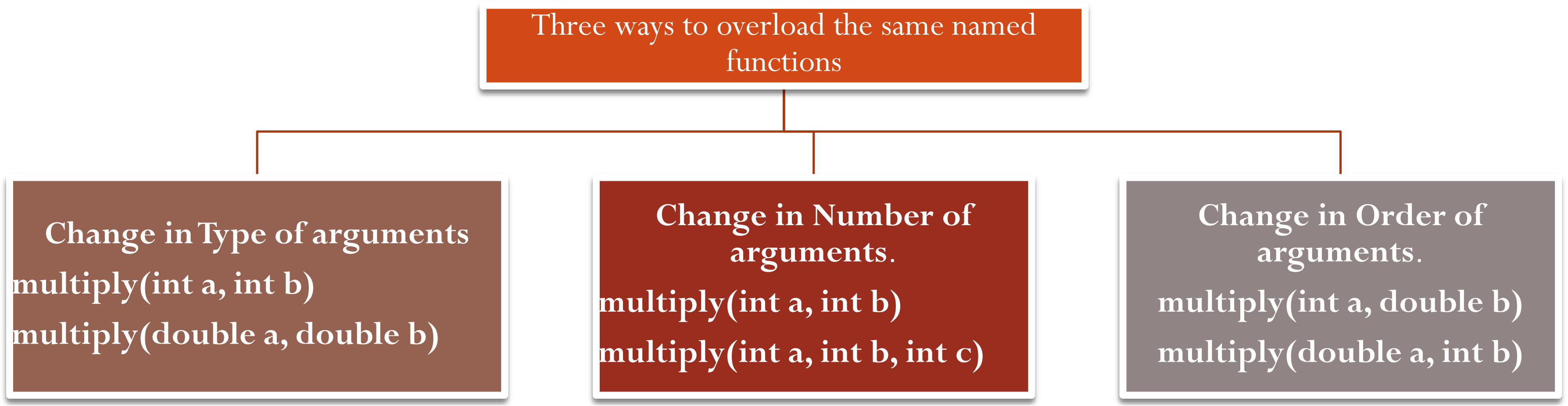
Run Time

Achieved by achieved by function Overriding

Overloading and Overriding in Java for Selenium



Method Overloading in Java for Selenium


Three ways to overload the same named functions
Change in Type of arguments
multiply(int a, int b)
multiply(double a, double b)
Change in Number of arguments.
multiply(int a, int b)
multiply(int a, int b, int c)
Change in Order of arguments.
multiply(int a, double b)
multiply(double a, int b)

Method Overloading - Demo

Method Overriding or Runtime Polymorphism in Java for Selenium

- When a derived class has a definition for one of the member functions of the base class

- Base function is said to be **overridden**.

Method Overriding or Runtime Polymorphism in Java for Selenium

```
class Parent {
  void Print()
   {
     System.out.println("parent class");
   }
}
```

```
class subclass1 extends Parent {
   void Print()
   {
     System.out.println("subclass1");
   }
}
```

```
class subclass2 extends Parent {

   void Print()
   {
     System.out.println("subclass2");
   }
}
```

```
class TestPolymorphism3 {
   public static void main(String[] args)
   {

      Parent a;

      a = new subclass1();
      a.Print();

      a = new subclass2();
      a.Print();
   }
}
```

Method Overriding or Runtime Polymorphism - Demo

Abstraction in Java for Selenium

Abstraction in java for selenium

- **abstraction** is the process of hiding certain details and showing only essential information to the use
- Can be achieved with either **abstract classes** or **interfaces**
- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

```
abstract class Animal {
public abstract void animalSound();

 public void sleep() { System.out.println("Zzz"); } }

public static void main(String[] args) {
  Animal myObj = new Animal();
}
```

This line will
Generate Error

Abstraction Demo

Interface in Java for Selenium

Interface

- Used for full abstraction

- The class that implements interface must implement all the methods of that interface

- Interface uses the implements keyword

- You to extend more than one class However you can implement more than one interfaces in your class.

Interface Demo

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
 	
public abstract void draw();
 	
}
	Example:
	public interface Drawable{
	
	void draw();
	
	}

Access modifiers and Encapsulation in Java for Selenium

Access Modifiers in java for selenium

- **Access Modifiers** - controls the access level
- **Class Level Modifiers**

Modifier	**Description**
public	The class is accessible by any other class
default	The class is only accessible by classes in the same package. This is used when you don't specify a modifier.

Access Modifiers in java for selenium

- **Access Modifiers** - controls the access level
- **Attributes, methods and constructors Level Modifiers**

Modifier	Description
public	The code is accessible for all classes
private	The code is only accessible within the declared class
default	The code is only accessible in the same package. This is used when you don't specify a modifier.
protected	The code is accessible in the same package and subclasses.

Access Modifiers in java for selenium

	Within Same Class	Within same package	Outside the package- (Subclass)	Outside the package- (Global)
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes (only to derrived class)	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

Encapsulation in java for selenium

Encapsulation, is to make sure that "sensitive" data is hidden from users

declare class variables/attributes as private

provide public **GET** and **SET** methods to access and update the value of a private variable

-----

GET method returns the variable value, and SET   method sets the value.

Why Encapsulation in java for selenium

1. **Better control** of class attributes and methods
2. Class attributes can be made **read-only** or **write only**
3. **Flexible:** the programmer can change one part of the code without affecting other parts
4. Increased **security** of data

Constructer in Java for Selenium

Constructers

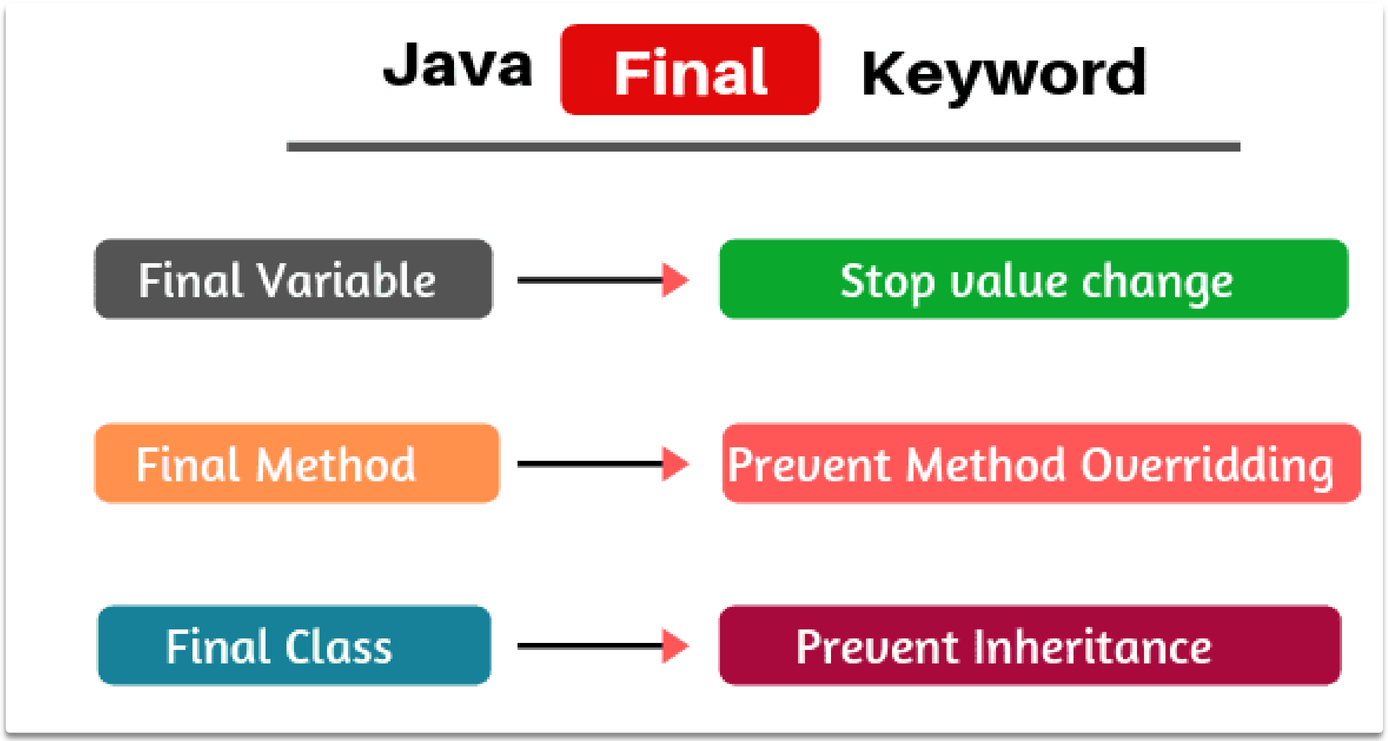
A constructor in Java is a **special method** that is used to initialize objects

- Constructer has same name as that of class
- The constructor is called when an object of a class is created.
- Every class has default constructer when no constructer is defined
- It can be used to set initial values for object variables
- There can be multiple constructers of the same class

Constructers Demo

Final keyword in java for selenium

Final

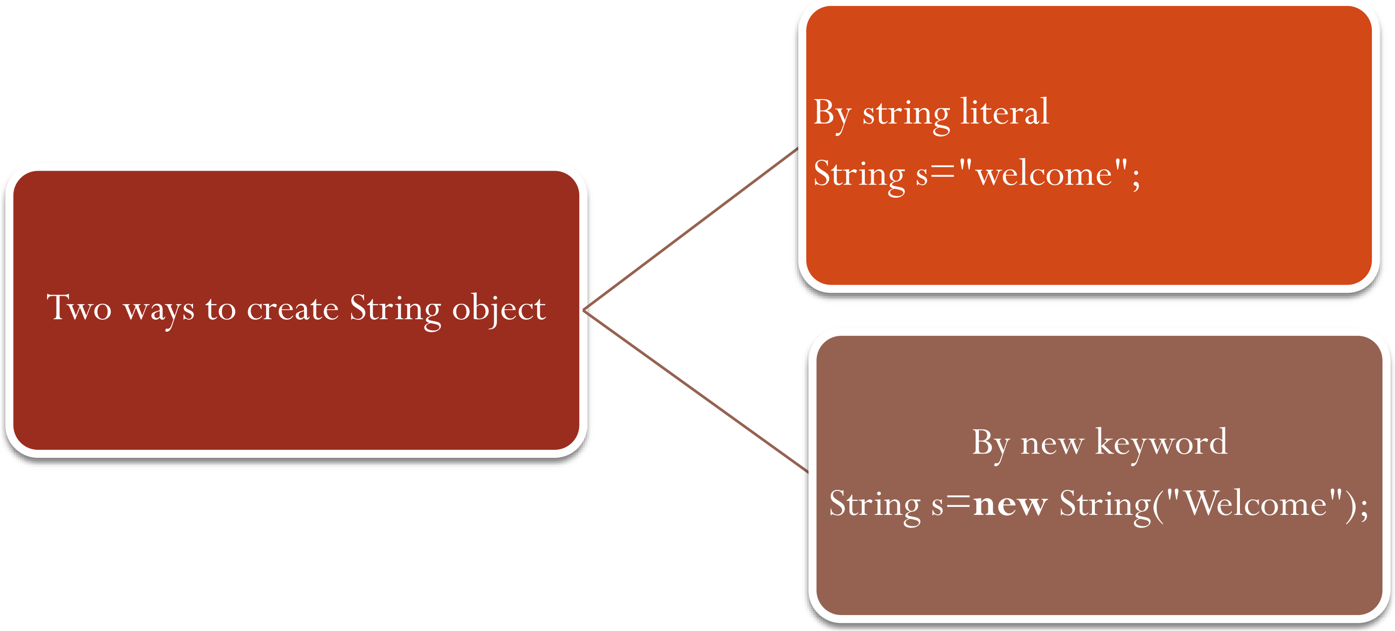

Java Final Keyword
Final Variable
Stop value change
Final Method
Prevent Method Overridding
Final Class
Prevent Inheritance

Final Demo

String Class in java for selenium

String in java for selenium

- String is a sequence of characters
- In Java, string is an object that represents a sequence of character
- java.lang.String class is used to create a string object



Two ways to create String object

By string literal

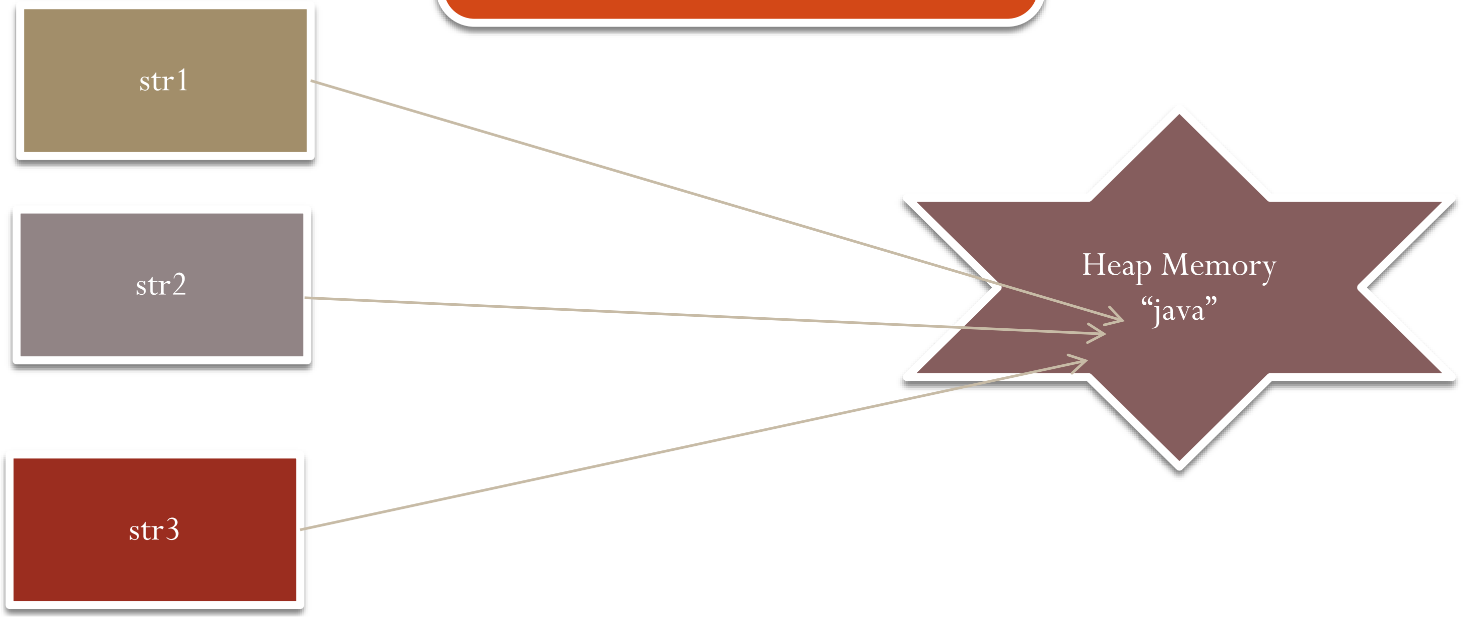
```
String s="welcome";
```

By new keyword

```
String s=new String("Welcome");
```

String Pool

```
String str1="java";
String str2="java";
String str3="java";
```



String Method

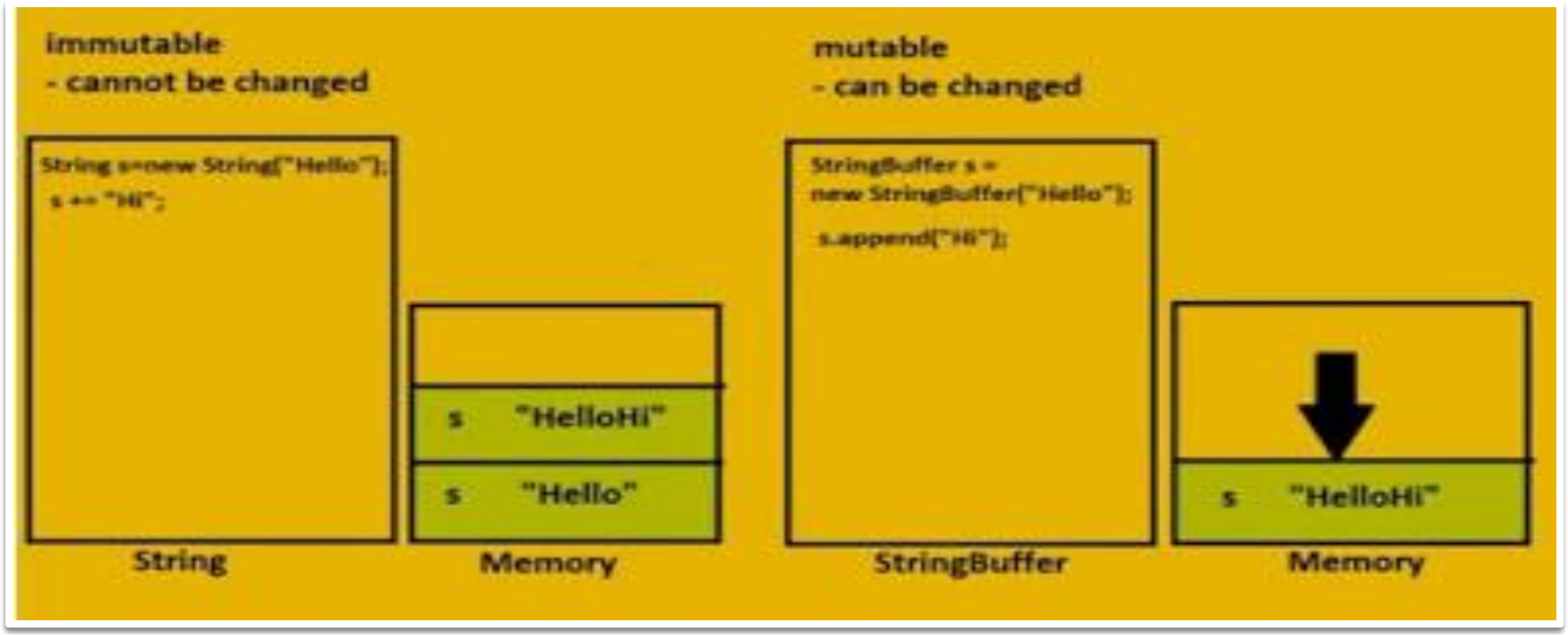
No.	Method	Description
1	**char charAt(int index)**	**returns char value for the particular index**
2	**int length()**	**returns string length**
3	static String format(String format, Object... args)	returns a formatted string.
4	static String format(Locale l, String format, Object... args)	returns formatted string with given locale.
5	**String substring(int beginIndex)**	**returns substring for given begin index.**
6	**String substring(int beginIndex, int endIndex)**	**returns substring for given begin index and end index.**
7	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string.
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	returns a joined string.
10	boolean equals(Object another)	checks the equality of string with the given object.
11	boolean isEmpty()	checks if string is empty.
12	**String concat(String str)**	**concatenates the specified string.**
13	String replace(char old, char new)	replaces all occurrences of the specified char value.
14	String replace(CharSequence old, CharSequence new)	replaces all occurrences of the specified CharSequence.

String Method

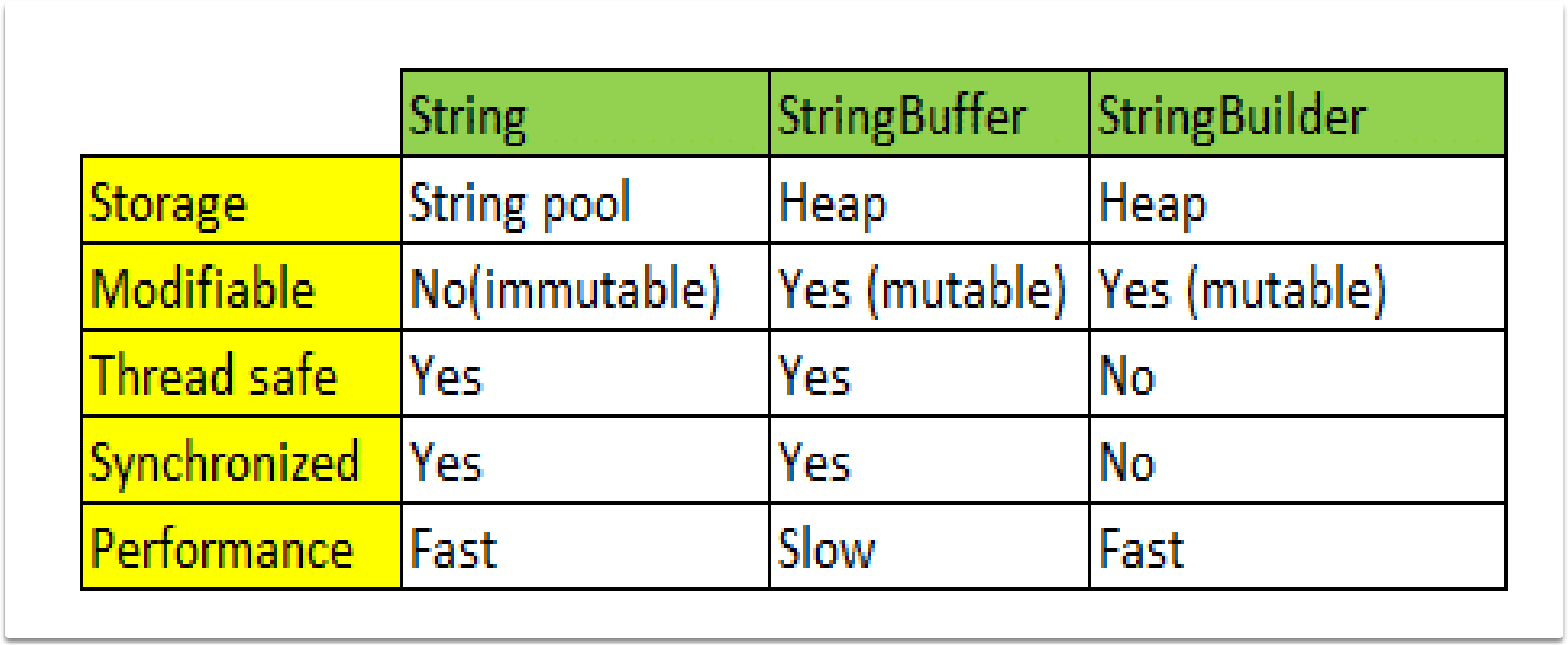
No.	Method	Description
15	**static String equalsIgnoreCase(String another)**	**compares another string. It doesn't check case.**
16	String[] split(String regex)	returns a split string matching regex.
17	String[] split(String regex, int limit)	returns a split string matching regex and limit.
18	String intern()	returns an interned string.
19	int indexOf(int ch)	returns the specified char value index.
20	int indexOf(int ch, int fromIndex)	returns the specified char value index starting with given index.
21	int indexOf(String substring)	returns the specified substring index.
22	int indexOf(String substring, int fromIndex)	returns the specified substring index starting with given index.
23	**String toLowerCase()**	**returns a string in lowercase.**
24	String toLowerCase(Locale l)	returns a string in lowercase using specified locale.
25	**String toUpperCase()**	**returns a string in uppercase.**
26	String toUpperCase(Locale l)	returns a string in uppercase using specified locale.
27	**String trim()**	**removes beginning and ending spaces of this string.**
28	static String valueOf(int value)	converts given type into string. It is an overloaded method

String Demo

String , String Buffer and String Builder



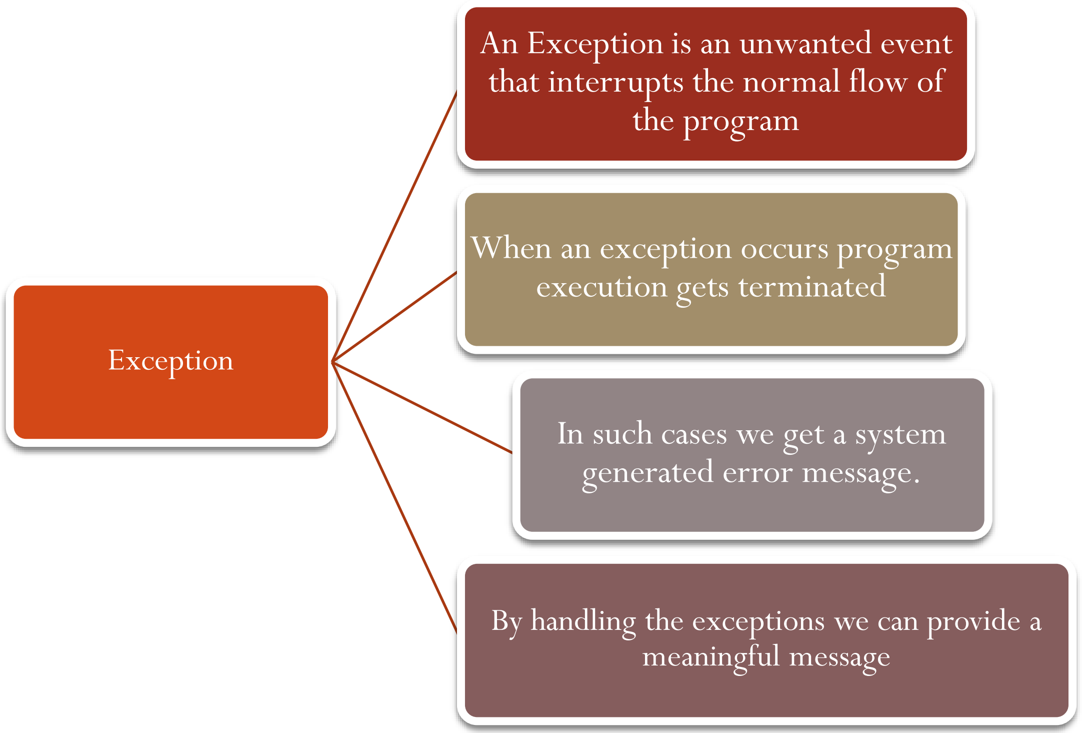
String vs String Buffer Vs String Builder



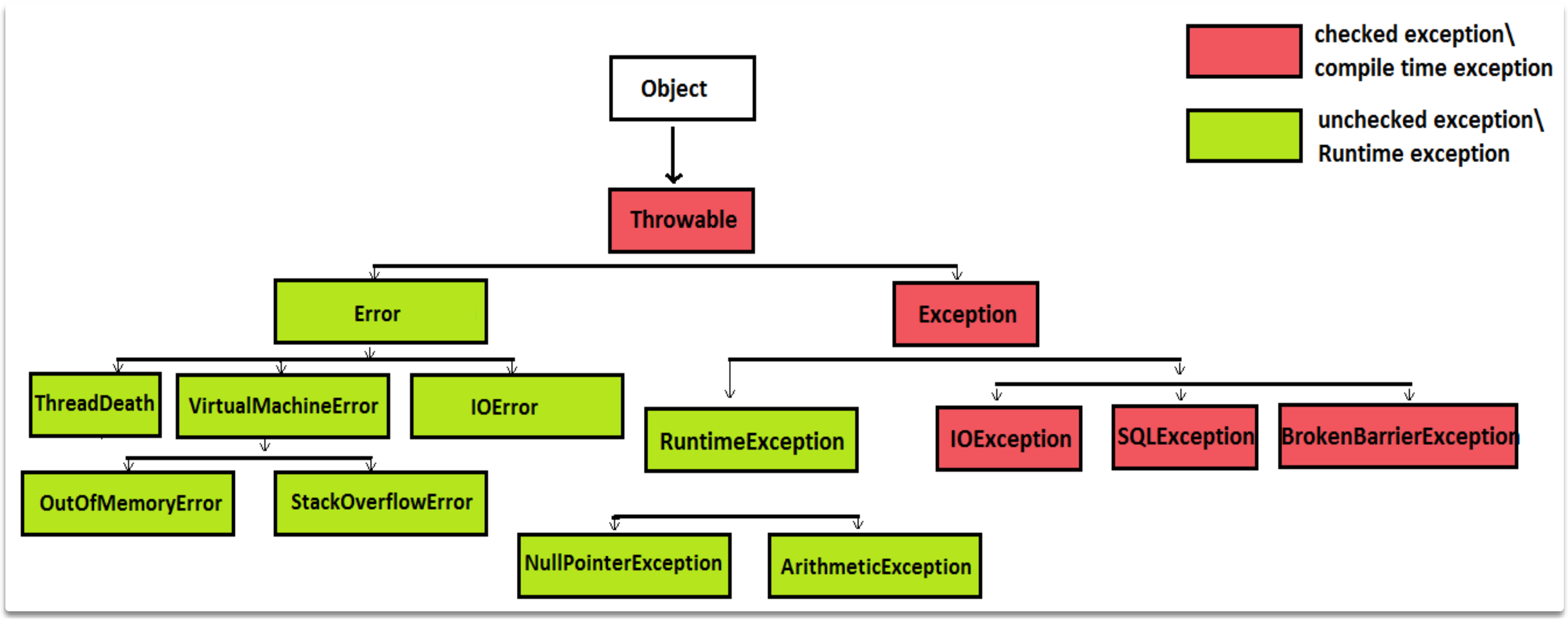
	String	StringBuffer	StringBuilder
Storage	String pool	Heap	Heap
Modifiable	No(immutable)	Yes (mutable)	Yes (mutable)
Thread safe	Yes	Yes	No
Synchronized	Yes	Yes	No
Performance	Fast	Slow	Fast

Exception Handling in Java for Selenium

Exception and Exception Handling


Exception
An Exception is an unwanted event that interrupts the normal flow of the program
When an exception occurs program execution gets terminated
In such cases we get a system generated error message.
By handling the exceptions we can provide a meaningful message

Types of Exception and Exception hierarchy in java



try-catch block syntax

```
  try {
//statements that may cause an exception
}
catch (exception(type) e(object))
{ //error handling code
}
```

- If no exception occurs in try block then the catch blocks are completely ignored
- There can be multiple catch blocks for one try block

Demo

finally block syntax

```
  try {

//statements that may cause an exception

}

catch (exception(type) e(object))
{
 //error handling code
}
finally
{
//Statements to be executed
}
```

- you cannot use finally without a try block.
- place those statements in this block that must be executed always
- Finally block is optional,
- An exception in the finally block, behaves exactly like any other exception

throw keyword

```
throw new ArithmeticException("dividing a number by 5 is not allowed in this program");
```

We can define our own set of conditions or rules and throw an exception explicitly using throw keyword

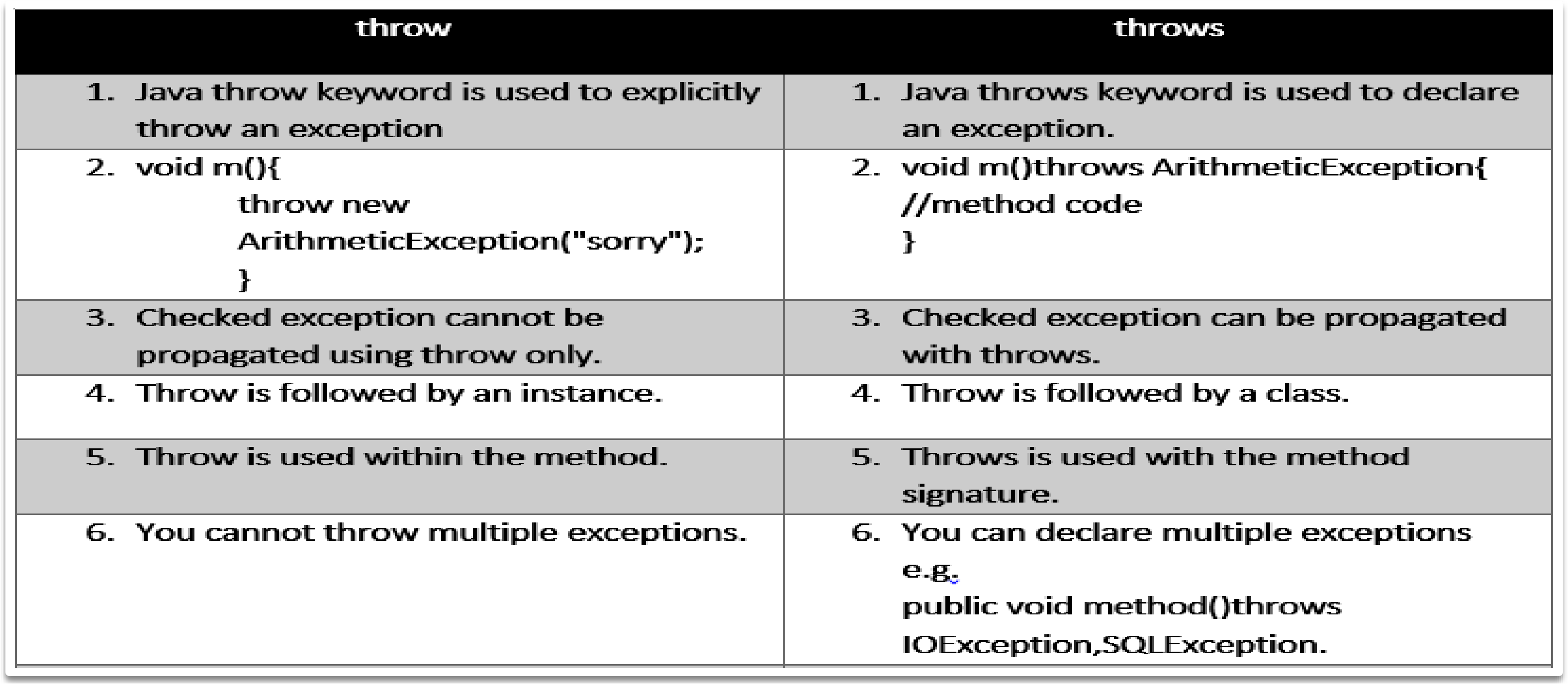
Demo User Defined Exception

throws keyword

```
class Test
{
   public static void main(String[] args)throws InterruptedException
   {
      Thread.sleep(10000);
      System.out.println("Hello Friends");
   }
}
```

- throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM)
- caller method is responsible to handle that exception.
- throws keyword is required only for checked exception and usage
- throws keyword for unchecked exception is meaningless.

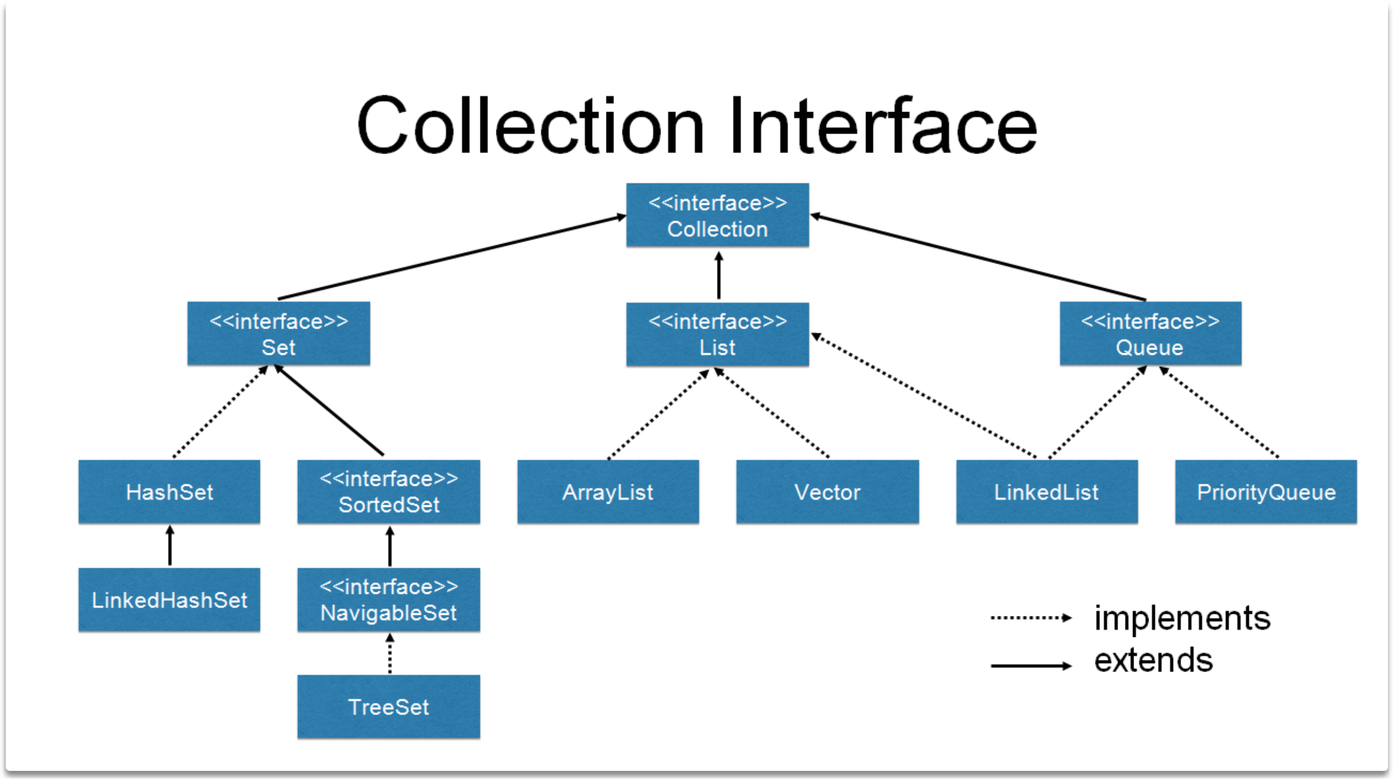
throws keyword



throw	throws
1. Java throw keyword is used to explicitly throw an exception	1. Java throws keyword is used to declare an exception.
2. void m(){ throw new ArithmeticException("sorry"); }	2. void m()throws ArithmeticException{ //method code }
3. Checked exception cannot be propagated using throw only.	3. Checked exception can be propagated with throws.
4. Throw is followed by an instance.	4. Throw is followed by a class.
5. Throw is used within the method.	5. Throws is used with the method signature.
6. You cannot throw multiple exceptions.	6. You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

Collection Framework in java – Array List , HashSet, HashMap for selenium

Collection Framework in java



Collection Framework – Array List

- The ArrayList class is a Dynamic resizable array
- Java ArrayList allows random access because array works at the index basis.
- It is in the package - java.util package
- Elements can be added and removed from an ArrayList
- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.

Array List Syntax

```
ArrayList<String> list=new ArrayList<String>();

list.add("Sandip");
list.add("Arohi");
list.add("Dipali");
 list.add("Atharva");

System.out.println(list);

 Iterator itr=list.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
```

Adding objects one by one and Invoking Entire Array list

Collection Framework – HashSet

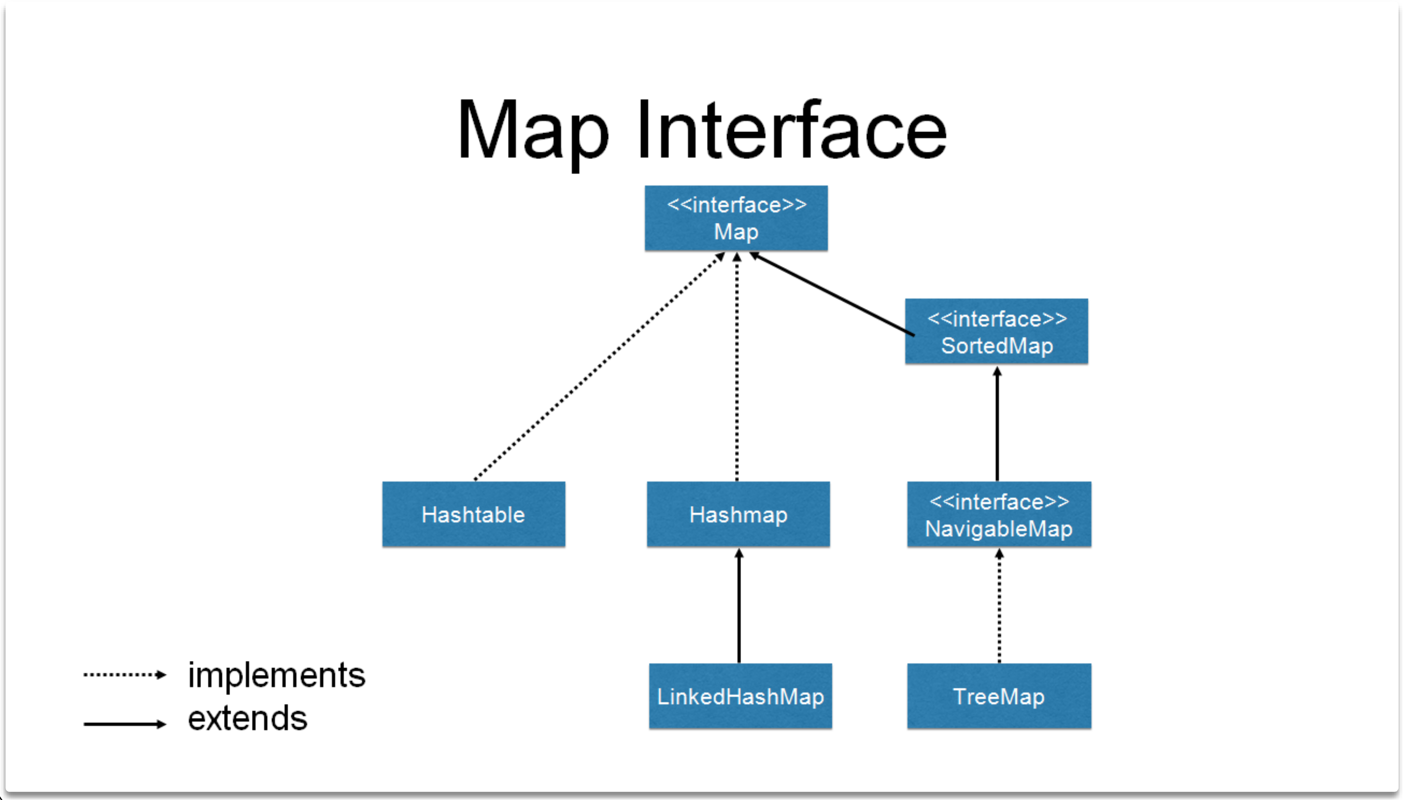
- HashSet stores the elements by using a mechanism called hashing.
- HashSet contains unique elements only.
- HashSet allows null value.
- HashSet class is non synchronized.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
- HashSet is the best approach for search operations.

Hash Set Syntax

```
HashSet<String> set=new  HashSet<String>();

set.add("Sandip");
set.add("Arohi");
set.add("Dipali");
 set.add("Atharva");

System.out.println(list);

 Iterator itr=set.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
```

Adding objects one by one and Invoking Entire Array list

Collection Framework –Map



Collection Framework – HashMap

- HashMap class contains values based on the key.
- HashMap class contains only unique keys.
- HashMap class may have one null key and multiple null values.
- HashMap class is non synchronized.
- HashMap class maintains no order.

HashMap

```
HashMap<Integer,String> hm=new HashMap<Integer,String>();

System.out.println("Initial list of elements: "+hm);
    hm.put(100,“Sandip");
    hm.put(101,“Arohi");
    hm.put(102,“Dipali");

    System.out.println("After invoking put() method ");

for(Map.Entry m:hm.entrySet()){
     System.out.println(m.getKey()+" "+m.getValue());
    }
```

Adding objects one by one and Iterating over the Hash Map

Session Content

01 Presentation

N/A

02 Exercise

Follow the demos in the videos and get some hands on with Eclipse

03 Notes

N/A

Recap

Methods , Classes and Object Introduction
Create class , Methods parameterized and non parameterized , with and without return type , Create object of class etc

Object oriented programming for selenium
Introduction to Opps and detail understanding of Inheritance,Polimorphism, Encapsulation , Abstraction

Final Keyword in Java
Using final for Class , Method , Variables

String , String Buffer and String Builder for selenium
String Class methods

Exception Handling in Java for Selenium
Checked and Unchecked Exception , Try, Catch , Finally , Throws , Throw etc

Collection Framework in java for selenium
Array List , Hash Set , Hash Map

Now

Next

01 **Selenium Automaton Beginner**
Introduction to Selenium WebDriver

Any questions?