# Java

## Different ways of creating object

* By new keyword
* By newInstance() method
* By clone() method
* By factory method etc.
* Anonymous object: If you have to use an object only once, annonymous object is a good approach.

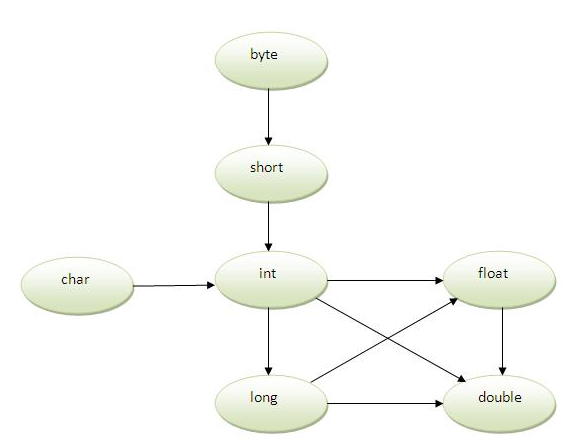
**new** Calculation().fact(5);

* Creating multiple objects by one type only

Rectangle r1=**new** Rectangle(),r2=**new** Rectangle();

## Method Overloading

* multiple methods by same name but different parameters.
* Compile time polymorphism.
* increases the readability of the program.
* two ways to overload the method in java: By changing number of arguments, By changing the data type
* Method overloading is not possible by changing the return type of the method because there may occur ambiguity.
* One type is promoted to another implicitly if no matching datatype is found.



* One type is not de-promoted implicitly for example double cannot be depromoted to any type implicitely.

## Constructor

1. Two types: Default constructor (no-arg constructor), Parameterized constructor

* Default constructor syntax:

<class\_name>(){}

* You can perform any operation in the constructor as you perform in the method like object creation, starting a thread, calling method etc.

### Copy Constructor

* There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.
* ways to copy the values of one object into another in java

1) By constructor

Student6(Student6 s){

id = s.id;

name =s.name;

}

2) By assigning the values of one object into another

3) By clone() method of Object class

## static keyword

* The static can be:

1) variable (also known as class variable): used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees

2) method (also known as class method)

3) block

4) nested class

### Restrictions for static method:

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

### When

Define static methods in the following scenarios only:

1. If you are writing utility classes and they are not supposed to be changed.
2. If the method is not using any instance variable.
3. If any operation is not dependent on instance creation.
4. If there is some code that can easily be shared by all the instance methods, extract that code into a static method.
5. If you are sure that the definition of the method will never be changed or overridden. As static methods can not be overridden.

### static block

* used to initialize the static data member
* executed before main method at the time of classloading

class A2{

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

public static void main(String args[]){

System.out.println("Hello main");

}

}

## this keyword

1. used to refer current class instance variable.

If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

**class** Student10{

**int** id;

    String name;

    Student10(**int** id,String name){

    id = id;

    name = name;

  }

The parameters to the constructor and the instance variables have the same name. So, to resolve the ambiguity use this. To reference instance variable.

Student11(**int** id,String name){

**this**.id = id;

**this**.name = name;

   }

1. this() can be used to invoke current class constructor.

 This approach is better if you have many constructors in the class and want to reuse that constructor. Call to this() must be the first statement in constructor.

 Student14(**int** id,String name){

**this**.id = id;

**this**.name = name;

    }

    Student14(**int** id,String name,String city){

**this**(id,name);//now no need to initialize id and name

**this**.city=city;

    }

1. this keyword can be used to invoke current class method (implicitly)
2. this can be passed as an argument in the method call.

It is mainly used in the event handling.

1. this can be passed as argument in the constructor call.
2. this keyword can also be used to return the current class instance.

## Inheritance

* Why?

1) For Method Overriding (so runtime polymorphism can be achieved).

2) For Code Reusability.

* Syntax

class Subclass-name extends Superclass-name

{

//methods and fields

}

* IS-A relationship
* Types: single, multilevel and hierarchical
* Multiple inheritance is not supported in java through class. Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

## Aggregation

* HAS-A relationship
* An object contains another object in it.

## Method Overriding

* If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.
* used for runtime polymorphism.
* Rules for Java Method Overriding

1) must have same name as in the parent class

2) method must have same parameter as in the parent class.

3) must be IS-A relationship (inheritance).

* Static method cannot be overridden because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.
* If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

## super keyword

* used to refer immediate parent class object.
* Usage

1) refer immediate parent class instance variable.

2) invoke immediate parent class constructor.

3) invoke immediate parent class method.

* super() is added in each class constructor automatically by compiler.

## Instance initializer block

1. The instance initializer block is created when instance of the class is created.
2. The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
3. The instance initializer block comes in the order in which they appear.

## Final Keyword

* Final can be:

1. variable
2. method
3. class

* A final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.
* If you make any class as final, you cannot extend it.
* final method is inherited but you cannot override it.
* If you declare any parameter as final, you cannot change the value of it.

## Polymorphism

* single action by different ways
* Two types of polymorphism in java: compile time polymorphism and runtime polymorphism.

### Upcasting

* reference variable of Parent class refers to the object of Child class
* Method is overridden not the datamembers, so runtime polymorphism can't be achieved by data members.

## Static Binding and Dynamic Binding

* Connecting a method call to the method body is known as binding.
* two types of binding

1. static binding (also known as early binding).
2. dynamic binding (also known as late binding).

### static binding

* type of the object is determined at compiled time(by the compiler)
* If there is any private, final or static method in a class, there is static binding

### Dynamic binding

* type of the object is determined at run-time

## instanceof

* Used to test whether the object is an instance of the specified type (class or subclass or interface).
* also known as type *comparison operator* because it compares the instance with type.
* Returns either true or false.
* If we apply the instanceof operator with any variable that has null value, it returns false.

## Abstraction

* Hiding the implementation details and showing only functionality to the user.
* two ways to achieve abstraction in java

1) Abstract class (0 to 100%)

2) Interface (100%)

* A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.
* A method that is declared as abstract and does not have implementation is known as abstract method.
* If there is any abstract method in a class, that class must be abstract.
* Rule: If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

## Interface

* Blueprint of a class. It has static constants and abstract methods only.
* also **represents IS-A relationship**.
* three reasons to use interface:

1) achieve fully abstraction.

2) support the functionality of multiple inheritance.

3) achieve loose coupling.



* The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.
* a class extends another class, an interface extends another interface but a **class implements an interface**.

### Multiple inheritance in Java by interface

* class implements multiple interfaces, or an interface extends multiple interfaces
* **class** A7 **implements** Printable,Showable{
* Even if both the interfaces have same methods, its implementation is provided by class TestTnterface1, so there is no ambiguity.

### marker or tagged interface

* An interface that have no member is known as marker or tagged interface.

### Nested Interface

* An interface i.e. declared within another interface or class.
* The nested interface must be referred by the outer interface or class. It can't be accessed directly.
* Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
* Nested interfaces are declared static implicitely.
* **interface** Showable{

**void** show();

**interface** Message{

**void** msg();

  }

}

**class** TestNestedInterface1 **implements** Showable.Message{

**public** **void** msg(){System.out.println("Hello nested interface");}

**public** **static** **void** main(String args[]){

  Showable.Message message=**new** TestNestedInterface1();//upcasting here

  message.msg();

 }

}

## Package

* group of similar types of classes, interfaces and sub-packages.
* Advantage of Java Package

1) used to categorize the classes and interfaces so that they can be easily maintained.

2) provides access protection.

3) removes naming collision

* If you use package.\* then all the classes and interfaces of this package will be accessible but not subpackages.

## Access and Non-access Modifiers

* two types of modifiers in java: **access modifiers** and **non-access modifiers**.
* 4 types of java access modifiers:

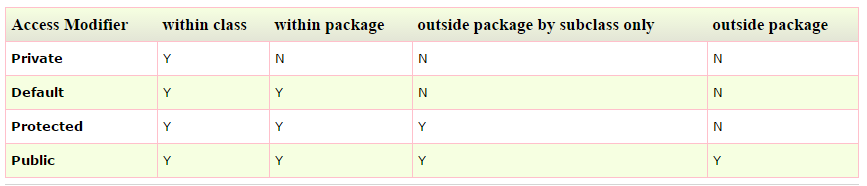
1) private

2) default

3) protected

4) public

* There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc.



### Access Modifiers

* If you make any class constructor private, you cannot create the instance of that class from outside the class.
* A class cannot be private or protected except nested class.

## Threads

There are two ways to create a thread:

1. By extending Thread class

class Multi extends Thread{

public void run(){

System.out.println("thread is running...");

}

public static void main(String args[]){

Multi t1=new Multi();

t1.start();

}

}

1. By implementing Runnable interface.

class Multi3 implements Runnable{

public void run(){

System.out.println("thread is running...");

}

public static void main(String args[]){

Multi3 m1=new Multi3();

Thread t1 =new Thread(m1);

t1.start();

}

}

### Commonly used Constructors of Thread class:

|  |
| --- |
| * Thread() * Thread(String name) * Thread(Runnable r) * Thread(Runnable r,String name) |

# ArrayList

 can contain duplicate elements.

 Java ArrayList class maintains insertion order.

 Java ArrayList class is non synchronized.

 Java ArrayList allows random access because array works at the index basis.

Two ways to iterate the elements of collection in java

1. By Iterator interface.
2. By for-each loop.

ArrayList<E>, on the other hand, allow fast random read access, so you can grab any element in constant time. But adding or removing from anywhere but the end requires shifting all the latter elements over, either to make an opening or fill the gap. Also, if you add more elements than the capacity of the underlying array, a new array (1.5 times the size) is allocated, and the old array is copied to the new one

Methods:

Add, addAll(Collection c),   al.removeAll(al2);  al.retainAll(al2);

# Linked List

LinkedList<E> allows for constant-time insertions or removals using iterators, but only sequential access of elements. In other words, you can walk the list forwards or backwards, but finding a position in the list takes time proportional to the size of the list. Javadoc says "operations that index into the list will traverse the list from the beginning or the end, whichever is closer"

LinkedList is faster in add and remove, but slower in get. Based on the complexity table and testing results, we can figure out when to use ArrayList or LinkedList.

In brief, LinkedList should be preferred if:

* there are no large number of random access of element
* there are a large number of add/remove operations

## Encapsulation

* *process of wrapping code and data together into a single unit*
* We can create a fully encapsulated class in java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

## Object class

* **Object class** is the parent class of all the classes in java by default.
* It is beneficial if you want to refer any object whose type you don't know.

## Object Cloning

* A way to create exact copy of an object.
* The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create.
* **class** Student18 **implements** Cloneable{

**int** rollno;

String name;

Student18(**int** rollno,String name){

**this**.rollno=rollno;

**this**.name=name;

}

**public** Object clone()**throws** CloneNotSupportedException{

**return** **super**.clone();

}

* Student18 s2=(Student18)s1.clone();
* clone() copies the values of an object to another. So we don't need to write explicit code to copy the value of an object to another.

## Array

* Collection of similar type of elements that have contiguous memory location.
* Advantages:

1) Code Optimization: It makes the code optimized, we can retrieve or sort the data easily.

2) Random access: We can get any data located at any index position.

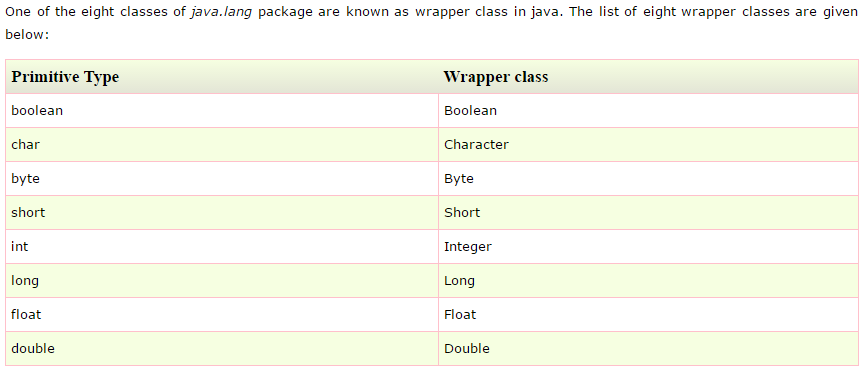
* Disadvantage: **Size Limit:** We can store only fixed size of elements in the array. It doesn't grow its size at runtime.
* two types of array: Single Dimensional Array, Multidimensional Array
* We can copy an array to another by the arraycopy method of System class.
* **public** **static** **void** arraycopy(

Object src, **int** srcPos,Object dest, **int** destPos, **int** length

)

* System.arraycopy(copyFrom, 2, copyTo, 0, 7);

## Wrapper class

* provides the mechanism to convert primitive into object and object into primitive.
* The automatic conversion of primitive into object is known and autoboxing and vice-versa unboxing.
* 
* **int** a=20;

Integer i=Integer.valueOf(a);//converting int into Integer

Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

* Integer a=**new** Integer(3);

**int** i=a.intValue();//converting Integer to int

**int** j=a;//unboxing, now compiler will write a.intValue() internally

## Call by Value and Call by Reference

* There is only call by value in java, not call by reference.
* If we pass object in place of any primitive value, original value will be changed.

## Strictfp Keyword

* strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable. The precision may differ from platform to platform that is why java programming language have provided the strictfp keyword, so that you get same result on every platform.
* **strictfp** **class** A{}//strictfp applied on class
* **strictfp** **interface** M{}//strictfp applied on interface
* **class** A{

**strictfp** **void** m(){}//strictfp applied on method

}

* The strictfp keyword **cannot** be applied on abstract methods, variables or constructors.

## String

* provides a lot of concepts that can be performed on a string such as compare, concat, equals, split, length, replace, compareTo, intern, substring etc.
* Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. String class is used to create string object.
* **char**[] ch={'j','a','v','a','t','p','o','i','n','t'};

String s=**new** String(ch);

* String is immutable i.e. it cannot be changed but a new instance is created. For mutable class, you can use StringBuffer and StringBuilder class.
* two ways to create String object:

1. By string literal - String s="welcome";
2. By new keyword

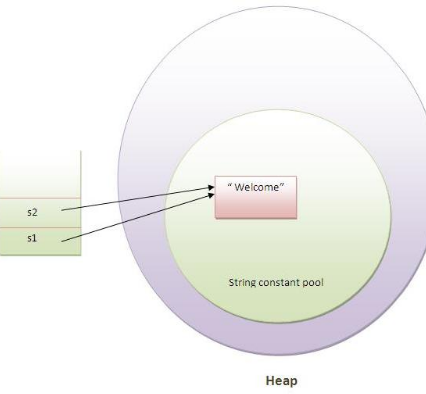
### Difference between string object and string literal

When you use a string literal the string can be [interned](http://en.wikipedia.org/wiki/String_interning), but when you use new String("...") you get a new string object. interned String object means, that the character sequence "abcde" will be stored at a central place, and whenever the same literal "abcde" is used again, the JVM will not create a new String object but use the reference of the cached String.

* Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

String s1="Welcome";

String s2="Welcome";//will not create new instance

* 
* String objects are stored in a special memory area known as string constant pool.
* String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap(non pool).

* s.concat(" Tendulkar"); will not change the value of s.
* s=s.concat(" Tendulkar");   will concat Tendulkar to s.
* Because java uses the concept of string literal.Suppose there are 5 reference variables,all referes to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

### String compare

There are three ways to compare string in java:

1. By **equals()** method - compares the original content of the string

two methods:

* public boolean equals(Object another) compares this string to the specified object.
* public boolean equalsIgnoreCase(String another) compares this String to another string, ignoring case.

1. By = = operator - compares references not values

String s1="Sachin";

   String s2="Sachin";

   String s3=**new** String("Sachin");

   System.out.println(s1==s2);//true (because both refer to same instance)

   System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)

1. By compareTo() method - compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

s1 == s2 :0

s1 > s2 :positive value

s1 < s2 :negative value

 String s1="Sachin";

String s2="Sachin";

   String s3="Ratan";

   System.out.println(s1.compareTo(s2));//0

   System.out.println(s1.compareTo(s3));//1(because s1>s3)

   System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )

### Concatenation

* two ways:

1. By **+ (string concatenation)** operator
2. By **concat()** method

### Substring

* startIndex is inclusive and endIndex is exclusive.
* two methods:

1. **public String substring(int startIndex)**
2. **public String substring(int startIndex, int endIndex)**

### toUpperCase() and toLowerCase() method

1. String s="Sachin";
2. System.out.println(s.toUpperCase());//SACHIN
3. System.out.println(s.toLowerCase());//sachin
4. System.out.println(s);//Sachin(no change in original)

### trim()

eliminates white spaces before and after string.

### startsWith() and endsWith()

1. String s="Sachin";
2. System.out.println(s.startsWith("Sa"));//true
3. System.out.println(s.endsWith("n"));//true

### charAt()

* returns a character at specified index.

### length()

returns length of the string.

### String format

1. **public** **static** String format(String format, Object... args)
2. and,
3. **public** **static** String format(Locale locale, String format, Object... args)
4. String name="sonoo";
5. String sf1=String.format("name is %s",name);
6. String sf2=String.format("value is %f",32.33434);
7. String sf3=String.format("value is %32.12f",32.33434);//returns 12 char fractional part filling with 0

### intern()

A pool of strings, initially empty, is maintained privately by the class String.

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

### valueOf()

* coverts given type such as int, long, float, double, boolean, char and char array into string.

### replace()

* replaces all occurrence of first sequence of character with second sequence of character.

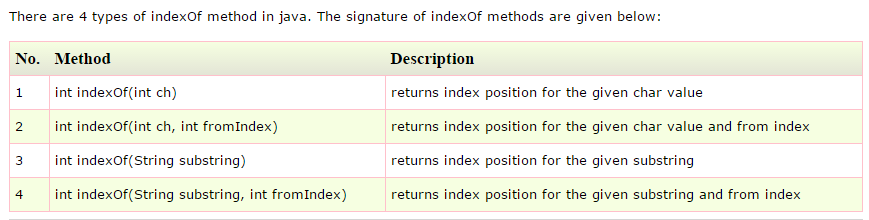
### getBytes()

returns sequence of bytes.

### getChars()

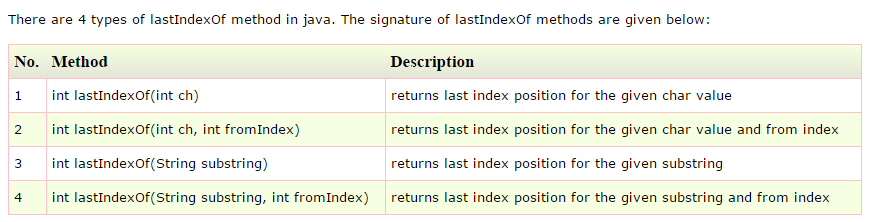
* copies the content of this string into specified char array.
* **public** **void** getChars(**int** srcBeginIndex, **int** srcEndIndex, **char**[] destination, **int** dstBeginIndex)

### indexOf

* returns index of given character value or substring. If it is not found, it returns -1.
* 

### lastIndexOf

returns last index of the given character value or substring. If it is not found, it returns -1.



### Intern

 returns the interned string. It can be used to return string from pool memory, if it is created by new keyword.

1. String s1=**new** String("hello");
2. String s2="hello";
3. String s3=s1.intern();//returns string from pool, now it will be same as s2
4. System.out.println(s1==s2);//false because reference is different
5. System.out.println(s2==s3);//true because reference is same

### isEmpty

checks if this string is empty. It returns *true*, if length of string is 0 otherwise *false*.

### Join

* returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.
* two types of join() methods:

**public** **static** String join(CharSequence delimiter, CharSequence... elements)

and

**public** **static** String join(CharSequence delimiter, Iterable<? **extends** CharSequence> eleents)

* String joinString1=String.join("-","welcome","to","javatpoint");

### Length

### Split(delimiter)

### StringBuffer

Constructors

1. **StringBuffer():** creates an empty string buffer with the initial capacity of 16.
2. **StringBuffer(String str):** creates a string buffer with the specified string.
3. **StringBuffer(int capacity):** creates an empty string buffer with the specified capacity as length.

**public synchronized StringBuffer append(String s):** is used to append the specified string with this string.

**public synchronized StringBuffer insert(int offset, String s) – inserts string at specified position.** The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

**public synchronized StringBuffer replace(int startIndex, int endIndex, String str)**

**public synchronized StringBuffer delete(int startIndex, int endIndex)**

**public synchronized StringBuffer reverse()**

**public int capacity() – returns current capacity.**  The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

**public void ensureCapacity(int minimumCapacity):** is used to ensure the capacity at least equal to the given minimum.  If min capacity is greater than the current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

**public char charAt(int index):**

**public int length()**

**public String substring(int beginIndex)**

**public String substring(int beginIndex, int endIndex)**

1. StringBuffer sb=**new** StringBuffer("Hello ");
2. sb.append("Java");//now original string is changed
3. System.out.println(sb);//prints Hello Java

### StringBuilder

StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

Constructors

1. **StringBuilder():** creates an empty string Builder with the initial capacity of 16.
2. **StringBuilder(String str):** creates a string Builder with the specified string.
3. **StringBuilder(int length):** creates an empty string Builder with the specified capacity as length.

public StringBuilder append(String s)

public StringBuilder insert(int offset, String s)

public StringBuilder replace(int startIndex, int endIndex, String str)

public StringBuilder delete(int startIndex, int endIndex)

public StringBuilder reverse()

public int capacity()

public void ensureCapacity(int minimumCapacity)

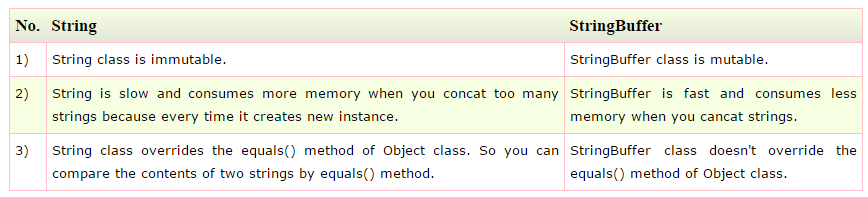
public char charAt(int index)

public int length()

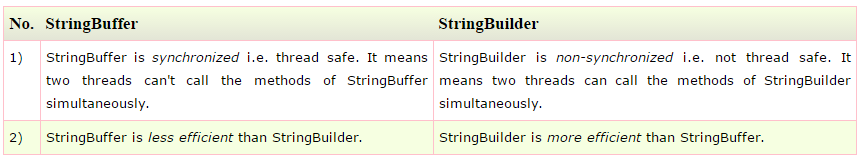
public String substring(int beginIndex)

public String substring(int beginIndex, int endIndex)

### String vs StringBuffer



### StringBuffer vs StringBuilder



## How to create Immutable class?

We can also create immutable class by creating final class that have final data members

1. **public** **final** **class** Employee{
2. **final** String pancardNumber;
4. **public** Employee(String pancardNumber){
5. **this**.pancardNumber=pancardNumber;
6. }
8. **public** String getPancardNumber(){
9. **return** pancardNumber;
10. }
12. }

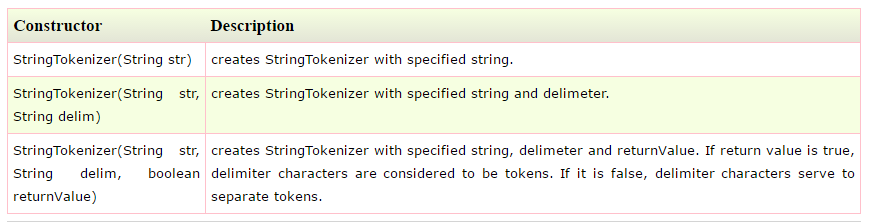
The above class is immutable because:

* The instance variable of the class is final i.e. we cannot change the value of it after creating an object.
* The class is final so we cannot create the subclass.
* There is no setter methods i.e. we have no option to change the value of the instance variable.

## toString()

* returns the string representation of the object.
* If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.
* If we don’t override the toString() method, hashcode values of the objects will be printed.
* **public** String toString(){//overriding the toString() method
* **return** rollno+" "+name+" "+city;
* }

## StringTokenizer Class

* allows you to break a string into tokens.
* 
* 
* StringTokenizer class is deprecated now. It is recommended to use split() method of String class or regex (Regular Expression).

## Finalize()

The method is called by the garbage collector when it determines no more references to the object exist.

The finalize method is called when an object is about to get garbage collected. That can be at any time after it has become eligible for garbage collection.

# Python

import unittest

class TestArithmetic(unittest.TestCase):

    def test\_addition(self):

        a = 100

        b = 200

        assert a+b == 3000

def test\_subtraction(self):

        a = 100

        b = 200

        assert a-b == 1000

def multiplication\_test(self):

        a = 100

        b = 200

        assert a\*b == 20000

[8:37:12 PM] Susmitha Reddy: class Parent(object):

reference = 1

class FirstChild(Parent):

pass

class SecondChild(Parent):

pass

[8:37:50 PM] Susmitha Reddy: print(Parent.reference, FirstChild.reference, SecondChild.reference)

Ans: **(1,1,1)**

[8:38:58 PM] Susmitha Reddy: FirstChild.reference = 1000

**(1,1000,1)**

[8:39:36 PM] Susmitha Reddy: Parent.reference = 1111

**(1111, 1000, 1111)**

[8:42:49 PM] Susmitha Reddy: import unittest

class TestArithmetic(unittest.TestCase):

def test\_addition(self):

a = 100

b = 200

assert a+b = 3000

def test\_subtraction(self):

a = 100

b = 200

assert a-b = 1000

def multiplication\_test(self):

a = 100

b = 200

assert a\*b = 20000

[8:56:05 PM] Susmitha Reddy: reference\_list = ["Elvis", "Aaron", "Presley"]

[8:56:58 PM] pravalika b: reference-list.reverse90

[8:57:12 PM] pravalika b: print reference-list

[8:57:35 PM] Susmitha Reddy: class Connector(object):

def set\_port(self, port):

[8:59:27 PM] pravalika b: isinstance(port, int0

[9:00:06 PM] pravalika b: type(port0 == int

[9:01:00 PM] pravalika b: try:

except exception:

else;

[9:01:29 PM] Susmitha Reddy: first = 10

second = 20

[9:01:49 PM] Susmitha Reddy: 1020

[9:01:56 PM] pravalika b: print first, second

[9:03:56 PM] pravalika b: print first + str(second)

* **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
* **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

## Packages required

virtualenv

pip install selenium==2.40

pip install nose

## Sample Code

import unittest

from selenium import webdriver

class AweberTest(unittest.TestCase):

@classmethod

def setUpClass(cls):

cls.driver = webdriver.Firefox()

def test\_title(self):

self.driver.get('https://www.aweber.com')

self.assertEqual(

self.driver.title,

'AWeber Email Marketing Services & Software Solutions for Small Business')

@classmethod

def tearDownClass(cls):

cls.driver.quit()

## Find Element

self.driver.get('https://www.aweber.com')

order\_tab = self.driver.find\_element\_by\_css\_selector('#ordertab>a')

order\_tab.click()

## List of elements

pricings = self.driver.find\_elements\_by\_css\_selector('#pricing-plans .price')

pricing\_texts = [price.text for price in pricings]

## Assertions

self.assertEqual(

self.driver.title,

'AWeber Email Marketing Services & Software Solutions for Small Business')

self.assertIn('19', pricing\_texts)

 self.assertTrue(monthly\_radio.is\_selected())

self.assertFalse('Foo'.isupper())

## Pass

The **pass** statement in **Python** is used when a statement is required syntactically but you do not want any command or code to execute

## Reverse function of list

reference\_list = ["Elvis", "Aaron", "Presley"]

reference\_list.reverse()

print reference\_list

O/P: ['Presley', 'Aaron', 'Elvis']

## Str()

# Backend testing

**Some of the Backend Testing elements are:**

* Database
* APIs
* Servers

## Backend Database Testing

**Databases are usually validated for:**

* ACID properties
* CRUD operations
* Schema
* Migration
* Business rule conformance
* Security
* Performance

### Why do we test a database?

#### Data Mapping

* To check whether the fields in the UI/Front end forms and mapped consistently with the corresponding DB table (and also the fields within).  Typically this mapping information is defined in the requirements documents.
* Whenever a certain action is performed in the front end of an application, a corresponding CRUD (Create, Retrieve, Update and delete) action gets invoked at the back end. A tester will have to check if the right action is invoked and the invoked action in itself is successful or not.

#### **ACID properties validation**

* **Atomicity** means that a transaction either fails or passes. This means that even if a single part of transaction fails- it means that the entire transaction has failed. Usually this is called the “all-or nothing” rule.
* **Consistency**: A transaction will always result in a valid state of the DB
* **Isolation**: If there are multiple transactions and they are executed all at once, the result/state of the DB should be the same as if they were executed one after the other.
* **Durability**: Once a transaction is done and committed, no external factors like power loss or crash should be able to change it

#### Data integrity

This means that following any of the [CRUD operations](http://en.wikipedia.org/wiki/Create,_read,_update_and_delete), the updated and most recent values/Status of shared data should appear on all the forms and screens. A value should not be updated on one screen and display an older value on another one. So devise your DB test cases in a way to include checking the data in all the places it appears to see if it is consistently the same.

#### Business rule conformity

More complex databases means more complicated components like relational constraints, triggers, stored procedures, etc. So testers will have to come up with appropriate SQL queries in order to validate these complex objects.

### SQL commands

The most commonly used command is the “Select”.

*Select \* from <tablename> where <condition>*

Apart from select, SQL has 3 important types of commands:

1. DDL : Data definition language
2. DML: Data manipulation language
3. DCL: data control language

#### Data Definition language:

Uses CREATE, ALTER, RENAME, DROP and TRUNCATE to handle tables (and indexes).

* Create table: *Create table<tablename> (field1 datatype(field size) ,……………..fieldn datatype(field size))*
* Delete entire table*: Drop table <tablename>.* – this command cannot be rolled back

#### Data Manipulation language:

Includes statements to add, update and delete records.

* To insert a row into a DB: INSERT INTO <table name> (field1, field2, field3)  VALUES  (‘val1’, ‘val2’…’valn’);
* Delete specific row/rows from a table: *DELETE FROM TABLENAME WHERE <required condition>.*
* Update rows: *UPDATE <tablename> SET field1 = ‘updated value’ WHERE field2 = ‘N’;*

#### Data control language:

Deals with giving the authorization to users for manipulation and access to the data. Grant and Revoke are the two statements used.

Grant syntax:  
*Grant select/update*  
*On <table name>*  
*To <user id1, user id2…useridn>;*

Revoke syntax:  
*Revokeselect/update*  
*on <table name>*  
*from<user id1, user id2…useridn>;*

### What to test – different components

#### 1) Transactions:

When testing transactions it is important to make sure that they satisfy the ACID properties.

The following are the statements commonly used:

* BEGIN TRANSACTION TRANSACTION#
* END TRANSACTION TRANSACTION#

Rollback statement ensures that the database lies in a consistent state.

* ROLLBACK TRANSACTION#

After these statements are executed, use a select to make sure if the changes have been reflected.

* SELECT \* FROM TABLENAME <tables which involve the transactions>

#### 2) Database schema:

Database schema is nothing but a formal definition of the how the data is going to be organized into a DB. To test it:

* Identify the requirements based on which the database operates. Sample requirements:
  + Primary keys to be created before any other fields are created.
  + Foreign keys should be completely indexed for easy retrieval and searching.
  + Field names starting or ending with certain characters.
  + Fields with a constraint that certain values can or cannot be inserted.
* Use one of the following ways according to the relevance:
  + SQL Query *DESC<table name>* to validate the schema.
  + Regular expressions for validating the names of the individual fields and their values
  + Tools like SchemaCrawler

#### 3) Trigger:

When a certain event takes places on a certain table, a piece of code (a trigger) can be auto-instructed to be executed.

**For example**, a new student joined a school. The student is taking 2 classes; math and science. The student is added to the “student table”.  A trigger could be adding the student to the corresponding subject tables once he is added to the student table.

The common method to test is to execute SQL query embedded in the trigger independently first and record the result. Follow this up with executing the trigger as a whole. Compare the results.

These are tested during both the black box and white box testing phases.

* **White box testing**:  Stubs and drivers are to insert or update or delete data that would result in the trigger being invoked. The basic idea is to just test the DB alone even before the integration with the front end (UI) is made.
* **Black box testing**:

**a)** Since the UI and DB integration is now available; we can insert/delete/update data from the front end in a way that the trigger gets invoked. Following that select statements can be used to retrieve the DB data to see if the trigger was successful in performing the intended operation.

**b)** Second way to test this is to directly load the data that would invoke the trigger and see if it works as intended.

#### 4) Stored Procedures:

Stored procedures are more or less similar to user defined functions. These can be invoked by a call procedure/execute procedure statements and the output is usually in the form of result sets.

These are stored in the RDBMS and are available for applications.

These are also tested during:

* **White box testing:** Stubs are used to invoke the stored procedures and then the results are validated against the expected values.
* **Black box testing:** Perform an operation from the frontend(UI) of the application and check for the execution of the stored procedure and its results.

#### 5. Field constraints – Default value, unique value and foreign key:

* Perform a front end operation which overruns the database object condition
* Validate the results with a SQL Query.

Checking the default value for a certain field is quite simple. It is a part of business rule validation. You can do it manually or you can use tools like QTP to do so. Manually, you can perform an action that will add a value other than the default value into the field from the front end and see if it results in an error.

**Primary Key**

* Primary key cannot have a NULL value.
* Each table can have only one primary key.
* By default, Primary key is clustered index and data in the database table is physically organized in the sequence of clustered index.
* Primary key can be related with another table's as a Foreign Key.
* We can generated ID automatically with the help of Auto Increment field. Primary key supports Auto Increment value.

**Unique Key**

* Unique Constraint may have a NULL value.
* Each table can have more than one Unique Constraint.
* By default, Unique key is a unique non-clustered index.
* Unique Constraint can not be related with another table's as a Foreign Key.
* Unique Constraint doesn't supports Auto Increment value.

**Foreign Key**

* Foreign key is a field in the table that is primary key in another table.
* Foreign key can accept multiple null value.
* Foreign key do not automatically create an index, clustered or non-clustered. You can manually create an index on foreign key.
* We can have more than one foreign key in a table.
* Foreign keys do not automatically create an index, clustered or non-clustered. You must manually create an index on foreign keys.
* There are actual advantages to having a foreign key be supported with a clustered index, but you get only one per table. What's the advantage? If you are selecting the parent plus all child records, you want the child records next to each other. This is easy to accomplish using a clustered index.
* Having a null foreign key is usually a bad idea instead of NULL  referred to as "orphan record"

**Sub query**

[Subquerie](http://www.essentialsql.com/get-ready-to-learn-sql-server-19-introduction-to-sub-queries/)s can be used to return either a scalar (single) value or a row set; whereas, joins are used to return rows.

* A common use for a subquery may be to calculate a summary value for use in a query.  For instance we can use a subquery to help us obtain all products have a greater than average product price.

**Temporary Tables**

great feature that lets you store and process intermediate results by using the same selection, update, and join capabilities that you can use with typical SQL Server tables.

they will be deleted when the current client session terminates.

SELECT TOP 2 \* FROM Customers;

SELECT TOP 50 PERCENT \* FROM Customers;

SELECT \* FROM TABLENAME ORDER BY random() LIMIT 5;

|  |  |
| --- | --- |
| % | A substitute for zero or more characters |
| \_ | A substitute for a single character |
| [*charlist*] | Sets and ranges of characters to match |
| [^*charlist*] or [!*charlist*] | Matches only a character NOT specified within the brackets |

logical operations in SQL have 3 possible values NOT 2 possible values. What are those 3 possible values? They are TRUE, FALSE, and UNKNOWN.

<http://www.programmerinterview.com/index.php/database-sql/what-is-an-index/>

### VBScript to validate data using Regex

<i>Function VBScriptRegularexpressionvlaidation(pattern , string\_to\_match)</i>

<i>Set newregexp = new RegExp</i>

<i>newregexp.Pattern = “<Default value as required by the business requirements>”</i>

<i>newregexp.Ignorecase = True</i>

<i>newregexp.Global = True</i>

<i>VBScriptRegularexpressionvlaidation = newregexp.Test(string\_to\_match)</i>

<i>End Function</i>

<i>Msgbox VBScriptRegularexpressionvlaidation(pattern , string\_to\_match)</i>

## DBfit

DBFit is a part of FitNesse, itself an implementation of the FIT acceptance testing framework. FIT concerns itself primarily with user acceptance testing rather than unit testing and FitNesse’s implmentation as a wiki allows functional users to get more directly involved in the testing of your application.

DBFit is directly targeted at database testing. Tests are written as wiki tables with a query or procedure execute followed by a listing of the expected results. It can handle direct SQL, stored procedures, or, with a bit of simple customization, custom procedures.

### Why DBfit?

### <http://blogs.perficient.com/dataanalytics/2012/06/07/bi-tools-testing-with-dbfit/>

<http://dbfit.github.io/dbfit/docs/reference.html>

# BDD and Cucumber

## TDD

Developer writes some unit tests first. They will first fail because nothing is implemented. Next the developer implements the code for the modules for which test cases are defined. If the code is implemented well the test cases will pass.

## BDD

Stakeholders define the behavior of application in simple english language and what they expect to see from the application. All coding efforts are geared towards delivering the desired behavior.

* Stakeholder/ non-technical person writes the features to be implemented.

Feature file – Gherkin

Step definition – using java or ruby

## TDD vs BDD

* If the implementation is not complete some of the test cases will fail. Developers understand the how much is pending but non technical people cannot. Whereas with BDD everyone can understand what has to be implemented and how much implementation is finished.
* language TDD – Programming lang used to define unit test cases. BDD – simple human understanding language like english

## Framework

<http://www.seleniumframework.com/cucumber-jvm-first-project/>

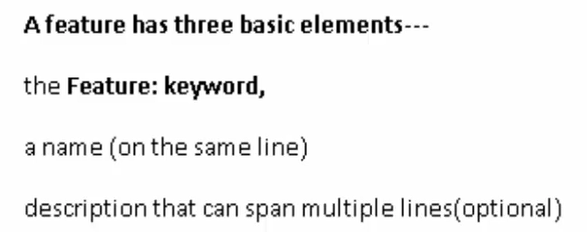
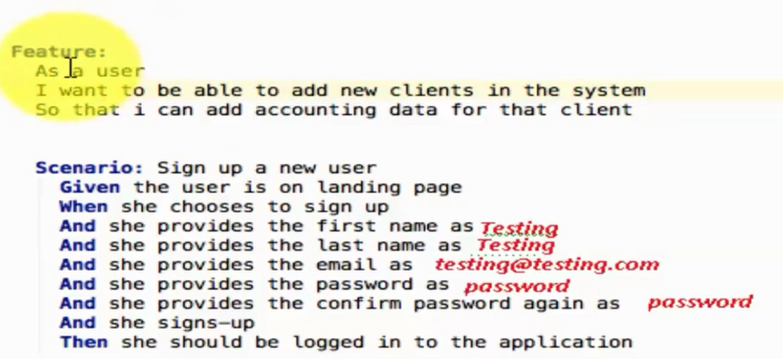
## Cucumber

* To automate acceptance tests written in BDD.
* Can be integrated with Selenium
* Two components: feature file and step definitions file

## Steps to create Cucumber Maven Junit Project

* Create a maven project
* Add dependencies for junit, cucumber junit and cucumber java
* Create a feature file under  src/test/resources

## Feature file

* We write acceptance scenario in simple english like language called Gherkin.
* 
* 

## Step definition file

* Test cases are written in Jave/ Ruby if Selenium is used, Ruby if Watir is used.

## testRunner

<http://toolsqa.com/cucumber/cucumber-options/>

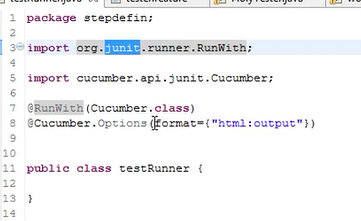
<https://sukesh15.gitbooks.io/cucumber-jvm-test-framework-/content/understanding_test_runners/index.html>

@RunWith(Cucumber.class)

@CucumberOptions (features = "src/test/java/features/", tags = "@Signup-DataDriven")

public class RunFeatures {

}



### testRunner with TestNG

package uk.co.automatictester.jwebfwk.runners;

import cucumber.api.CucumberOptions;

import cucumber.api.testng.AbstractTestNGCucumberTests;

@CucumberOptions(features = "src/test/resources/features/Download.feature",

        glue = "uk.co.automatictester.jwebfwk.glue",

        format = {"pretty"})

public class DownloadFeatureRunner extends AbstractTestNGCucumberTests {

}

## Gherkin/ Feature file

* Each line that is not blank has to start with a Gherkin keyword.
* Keywords:

Feature

Scenario

Given, When, Then, And, But (Steps)

Background

Scenario Outline

Examples

“””(Doc Strings)

|(Data Tables)

@(Tags)

* Feature is equivalent to Test Suite | Test Scenario (.feature)
* Each Feature can have multiple Test Cases (Scenario)
* Each test case can have multiple steps

#### Feature

* Feature: Login Functionality

Here we will test login by admin, super admin,

guest and customers

The highlighted text above is the description of the feature. It can span multiple lines.

#### Scenario

* Equivalent to acceptance test case
* Scenario: Login by Customer
* Scenario: Login by Guest
* Scenario: Login by Super Admin

#### Step

* Typically starts with Given, When or Then
* We can also use And or But

##### Given

Used to describe the initial context of the system like starting a browser and entering the URL.

There can be many Given Steps. We can to put an AND or BUT condition between each of them

Given User open browser

And Enter URL

##### When

Describes the actions performed

##### Then

Expected output

#### Background

This is used when same steps are repeated in the Given steps of all the scenarios in a feature file

Background:

Given user is on Login page

#### Step Arguments

Used to pass test data from feature file. Step Args are given in “”

#### Scenario Outline and Example

Scenario Outline used in the place of scenario when we want to perform data driven testing by using place holders in <>.

Example is used to define the data table. first row is the header of the table in this step. Headers names shud be same as those mentioned in <> in Scenario Outline

Scenario Outline: Login by Customer

When user enter username <username>

And user enter password <password>

And user click on Login Button

Then User should be logged in

Example:

|username|password|

|testing|abc@123|

|testing2|abc@1234|

|testing3|abc@1235|

#### Comment using #

#### Tags

Used to group scenarios

Prefixed with @

Any number of tags can be placed above feature, scenario, scenario outline or examples keywords.

All tags have to be separated with a space

When a tag is specified for a feature all the scenarios in the feature are executed

Similar to groups in TestNG

#### Strict option

**Strict:** if strict option is set to false then at execution time if cucumber encounters any undefined/pending steps then cucumber does not fail the execution and undefined steps are skipped and BUILD is SUCCESSFUL.

@RunWith(Cucumber.class)

@CucumberOptions(plugin = {"pretty"}, strict = false)

public class RunYoursTest

{

// This class will be empty

}

and if Strict option is set to true then at execution time if cucumber encounters any undefined/pending steps then cucumber does fails the execution and undefined steps are marked as fail and BUILD is FAILURE. This is what the Console output looks like:

@All

Feature: Login Functionality

Here we will test login by admin, super admin,

guest and customers

@Regression

Scenario: Login by Customer

Given User is on Login Page

When User enters username

And User enters password

And click on Login Button

Then User should be logged in

@Smoke @Regression

Scenario: Login by Guest

Given User open browser

And enters URL

When User clicks on Guest user link

And User enters email

And click on Submit

Then User should be logged in as a guest user

@Smoke

Scenario: Login by Super Admin

When user enter username “testing”

And user enter password “abc@123”

And click login button

Then User logged in successfully

## Step definition file

This file contains code corresponding to the feature file.

* Open command prompt and go to the folder where you want to create the project.
* Enter the command

Cucumber –-init

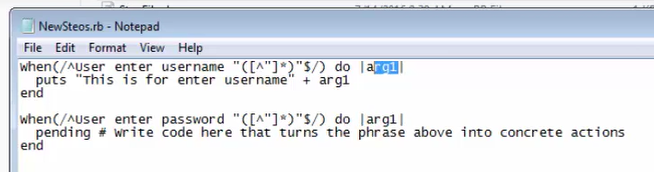
The above command generates a complete project structure with a features folder. This folder contains step\_definitions and support folder.

* Then create a feature file in features folder
* Enter the command

Cucumber

This shows the code snippet for undefined steps. And executes the code if steps are defined for all the scenarios

* Create step definition file with rb extension in step\_definitions folder and paste the above snippet
* If any arguments are passed thru feature file, they will be repersented as arg1 as shown below



## Hooks

They are placed in support folder. env.rb file is created by default. Any other file can also be created in this folder to create hooks.

### Before hook

* used to perform a task before each scenario

Before do

Puts “Before hook”

End

* To execute only before scenarios with a specific tag

Before(‘@<Tag name>’) do

Puts “Before”

End

* To execute a hook for scenarios which have any one of the mentioned tags

Before(‘@<Tag name1>, @<Tag name2>’) do

Puts “Before”

End

* To execute a hook for scenarios which have both the mentioned tags

Before(‘@<Tag name1>’, ‘@<Tag name2>’) do

Puts “Before”

End

### After hook

After do

Puts “After hook”

End

## Sample Hooks code

Require ‘selenium-webdriver’

Before do

Selenium::WebDriver::Chrome::Service.executable\_path=”<chrome driver path>”

$driver = Selenium::WebDriver.for :chrome #$driver is the global variable

:firefox

$driver.get “<URL>”

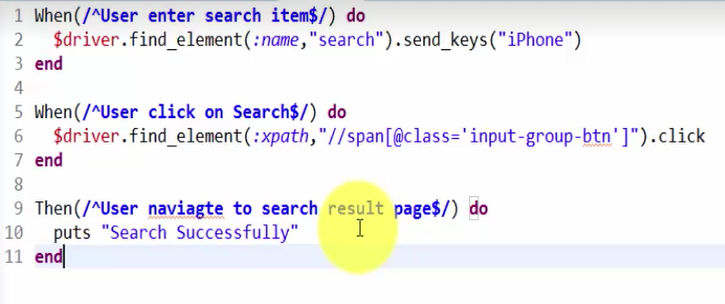
End

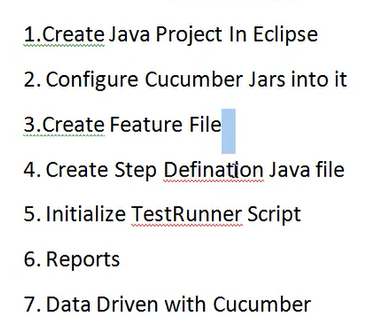
After

$driver.quit

end

## Sample step definition file

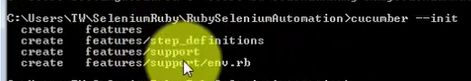




## Commands

### Cucumber –-init

Generates project structure



### Cucumber

Executes code and generates code snippet for undefined steps

### Cucumber –-tags @<tag name>

Executes only the scenarios with the given tag name

## Installation

Type the cmd

Gem install cucumber

To check if installed

Gem list –local

## Natural Plug in

Provides color coding and intellisense while writing feature

ansicon.exe – shows different colors for passed and failed tests Pass – green, Fail -red

## Ruby

Editors:

* Notepad – no intelisense or debug option
* Eclipse – Java is prerequisite for this. Also install Ruby plug in from eclipse marketplace then integrate ruby with eclipse by going to windows -> preferences -> Ruby -> Interpreter. Add the ruby.exe from the installed path

Type ruby –v command to see if ruby is installed. If so it will show the version

### Puts

Print smthng on console with line switch

### Print

Print smthng on console without line switc

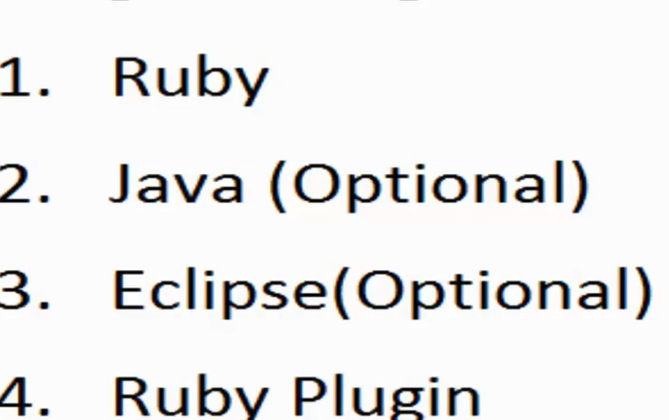
### Multiple line comments

=begin

Comment line1

Line2

=end



Class A1

Def initialize

Puts “this is a constructor”

end

Def methodName(a)

If(A%2==0)

Puts “Even”

Elsif(a<0)

Puts “Negative”

Else

Puts “Else”

End

End

Def sum(a,b)

C=a+b

Return c

End

End

In another file

Require ‘A1’

Obj= A1.new

Obj.methodName(10)

X=obj.sum(100,20)

### Class

Class A1

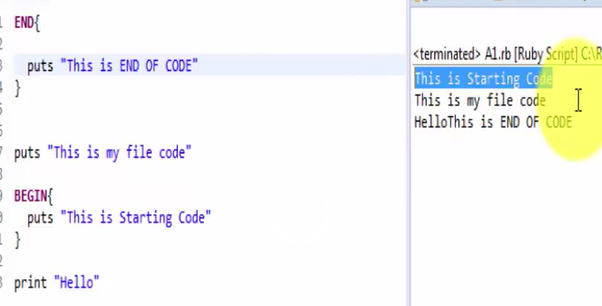
END

Starts with a capital letter

Ends with End keyword

Class members can be accessed by creating objects

### BEGIN and END block



### Constructor

Initialize method defined in the class

### While

I=1

While(i<=10)

--

I++

End

### For

For i in 1…10

End

… - excluding last value

.. – including last value

### Array

In ruby array can hold values of multiple data types

Arr1=Array.new(5)

Arr2=Array.new(2)

Arr1=[“Hello”, 1,2, “Three”,5]

Arr2[0]=”Hello”

Arr2[1]=0

TO access any element

Arr[0]

### ForEach

Use the above arr1

Arr1.each do |i|

--

End

### Hashes

Key value pair with any data type

Hash1 = Hash.new()

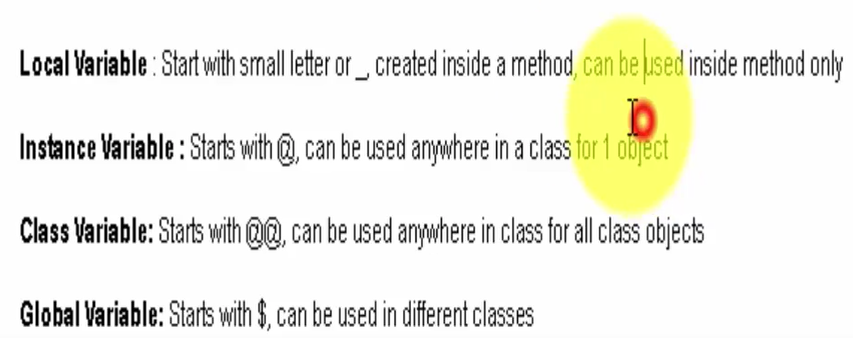
Hash1 = {“K1” => “Val1”, “K2” => 23 }

Hash[“K3”] = “New”

Hash1.length()

Hash1[“K1”]

### Type of variables



### Gem

Libraries in ruby similar to jar files in java

Gem install cucumber

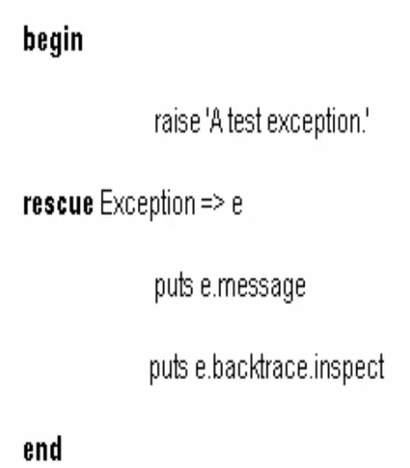
Gem list –local // lists all the gems installed

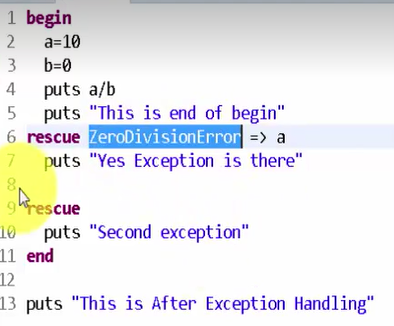
Gem uninstall cucumber

Gem fetch cucumber

Fetch just downloads the gem

### Exception handling





<http://minium.vilt.io/docs/load-external-data-cucumber/>