Polymorphism :

→ Change in the object behaviour in different stages of Appln development is called as polymorphism.

→ Polymorphism is one of the principle of object Oriented Programming.

→ polymorphism is a greek word which means many forms.

* There are 2 type of polymorphism
  1. Compile time polymorphism.
  2. Runtime polymorphism.

→ Based on method binding we can categorize polymorphism into two types.

→ Before any method executes each stack, binding should happens.

→ Method binding means referencing for two method calling.

→ In Compile time polymorphism, method binding happens at Compile time itself as Compiler will be having enough information to perform method binding.

→ We can achive Compile time polymorphism through Method overloading.

→ Compile time polymorphism is also called static polymorphism.

→ In Run time polymorphism method binding happens at Runtime.

→ Here, Compiler will not not have enough information to perform Method binding as method Overriding happens at Runtime.

→ To achive Runtime polymorphism, 3 things are Mandatory
  1. Overriding
  2. Inheritance
  3. Upcasting.

Abstraction

eg:

```
My Room
Static Switch mySwitch(char ch)
{ if (ch == 'B')
  {
    Bulb b = new Bulb();
    return new sub() }
  else if (ch == 'F')
  {
    return new Fan();
  else
    {return null;}
```

```
Switch
on() - void
off() - void
```

```
Fan (C)
on() - void
{ }
off() - void
{ }
```

```
Bulb (C)
on() - void
{ }
off() - void
{ }
```

```
Tester (C)
main()
{ Switch S1 = MyRoom.mySwitch('B');
  S1.on()
  S1.off()
Switch S2 = MyRoom.mySwitch('F')
  S2.on();
  S2.off();
}
```

* Abstraction is hidding the implemen and exposing the functionality or signature of a method is known as abstraction.

→ Abstraction is nothing but hidding the Complexity from the User and exposing the required functionality.

→ Abstraction is hiding the Class implementation from the usage.

→ we can define 100% abstraction by using

→ Interface which is 100% abstract.
→ While creating abstraction all the common features will be grouped in an interface and underlying class will override the same features based on the variations.

Packages :-

→ Package are nothing but folders in which we can group the similar kinds of programms.
→ Through package we can achive logical divisions while writing programs.
→ Through package we can achive encapsulation
→ Through package we can avoid naming collisions means we can have same names for classes, interfaces across different package.
→ Packages statement should be the first statement in a java file.

* Access levels / Access specifiers / privileges :-

→ Access levels are used to restrict on the access of the members
→ Through access levels we can achive encapsulation
→ There are four access levels :

Security ↓ / Visibility ↑

① Public
② Protected
③ default
④ Private

① Public :- Public member will have application level access i.e - public members can be accessed from all part of application.

(2) Default : Default members will have package level access i.e, Default members can be accessed only within the package in which it is developed.

(3) Private : Private members will have Class level access i.e private members can be accessed only within the Class in which it is declared in other words, private members cannot be accessed outside Class.

[NOTE] : (1) private member will not be inherited to subclass
(2) Private non-static methods cannot be override.

(4). Protected : Protected members will have package level access and with inheritance it can be accessed in other package.

** we should create subclass objects to access protected member not the parent class object.

⇒ If you want to access the members of a class which is in other package, we should first import that class by using import statement
e.g: import pack1.A;

→ Import statements should always be developed after package statements.

Pack-2

test()
test2()
test4()

→ Not Accessible

A a1 = new A()

test()
test2()

A

Public test1() - void
Protected test2() - void
test3() - void
Private test4 - void

* Encapsulation:

→ Restricting the direct access to data member and giving the indirect access to the data members through getters and setters method is called as encapsulation.

→ Here data member will be private and getters and setters method will be public.

→ Getters method will return the value of the private data member

→ Setters method will set the value of private data members.

→ Getters method are also called as accessors.

→ Setter method are also called as Mutators.

→ Through encapsulation, we will have complete control over the data which is getting manipulated

→ Through encapsulation, we can achieve security over the data.

→ Here the data and the code which will operate on the data will be binded together in the same class.

e.g.: email password.
bank balance.

Q.1-> What is a Java Bean Class? (EJB)

Ans: Developing a class with private data members and giving the indirect access to those private data members through getters and setters public service methods is called as a Java Bean Class.

Q.1 While method overriding can we change the access from default to public?

→ yes, while overriding we can expand the access level but we cannot weaken the access level if superclass method is default in subclass while overriding we can change it to protected or public.

But if super class method is public, we cannot change it to protected.

Q.2 Can we override private Non-Static Method?

Ans: private method cannot be overridden because private method will not inherit to the subclass and to achieve overridding inheritance is mandatory.

Q.3 Can we declare a class as protected or private

Ans: →No, protected and private are only for the members of the class not for the class itself. (it will be an error).

→ A class can be declared either public or default

→ public classes can be imported through import statement.

→ default classes cannot be imported.

⇒ Rule → In one Java file, we can develop maximum (0, or) one public class and that public class name should be the Java file name.

* Eclipse Rule: Main method should be present in the file class Name.

* Global Reference variable :

```
Public class A
{ static A as;
  int empid;
  static void test()
  { sop("from test :: " a1.empId); }
  psvm(String[] args)
  {
    sop(Main Starts
    a1 = new A1
    test();
  } sop("Main : :" + a1.empid")
```

```
main starts
from test :: 0
from main :: 0
Main Ends
```

Design Pattern :

① Singleton class - only one object

Tester (C)

```
main()
{ sop("Main Starts");
// A rv = new A(); // CTE
  A rv1 = A.demo();
  rv.m1();
A rv2 = A.demo();
  rv2.m1();
  A rv3 = A.demo();
  rv3.m1();
```

(A)

```
Static A as = new
Private A()
{ sop("Private Construction");
  void m1()
{ sop("from m1 method");
Static A demo()
{ if(as == null)
{ as = new A();
{ return a;
else { return a; }
```

Second Approach for Singleton class:

```
Tester (1)
Main()
{ sop (Main Start'
// B.rr = new B()
Brr1 = B.demo
{
```

```
(B)
{ static B b1 = new B();
  private B()
{ sop ("Private Constructor");}
  Public Static B demo()
  { return b1; }.
```

→ Designing the best solutions for frequently occurring problems and using the same solution hence forth if the problem is re-occurring it is called as design pattern.

① Singleton class design patterns:

→ The class for which we can create only one object is called as Singleton class.

→ Through Singleton design pattern we can manage the memory effectively.

→ To Achive Singleton behaviour

① The Constructor of the class should be private

② There should be one method which should return to single object of that Class.

② Immutable objects:

→ The objects for which states cannot be Changed is called as immutable object.

→ It is used in multithreading.

(E)

Tester

```
Main
{
 C rv = new (10, 10.5);
 Sop (rv.getA());
 Sop (rv.getD());
 Soplm("xxxxxxxx");
// rv.a = 90 // NO direct access
// rv.setA(90) // NO setters.
 C rv2 = new C(90, 90.5)
 Sop (rv2.getA());
 Sop (rv2.getD());
}
```

(C)

C

```
Private int a;
Private double d;
Public C(int a, double d)
{ this.a = a; this.d = d; }
Public int getA()
{ return a; }
Public double getB()
{ return d; }
```

* Making data members private and no setters method only getters.

Second approach for immutable object.

→ global final variable can be initilized through Constructor or Initialization.

Tester (C)

```
Main()
{
 D d1 = new D(10);
 Soplm(d1.a);
 D d2 = New D(90);
 Soplm(d2.a)
 D d3 = new D(190);
 Soplm(d3.a);
}
```

D

```
final int a;
Public D(int a)
{ this.a = a; }
```

10

90

190

* Object class :

→ Object class is a built in class and it is the supermost class for any class in Java.

→ Object Class, it is having def of non-static method, which will be inherited to every class of Java.

toString() hashcode() equals()	Non-final	getClass() clone	Used in reflection API
notify() notifyAll() wait()	final	finalize()	garbage Collection

[NOTE]: Used in multithreading for interthread communication.

→ The object class is available in built in package Java.lang
→ It is also imported.

Tester ①

```
main()
{ A rv1 = new A()
  Sop (rv1.toString())
  A rv2 = new A()
  Sop (rv2).
```

Tester ③

```
main()
{ C rv1 = new C();
  Sop (rv1);
  C rv2 = new C();
  Sop (rv2);
```

overriding to string method

```
Public String toString()
{
return
  "C class object";}
```

```
Test.c
main()
{
D rv1 = new D(10);
Sopln(rv1);  i value: 20
D rv2 = new D(20);
Sopln(rv2)   i value: 20
```

```
D
int i
(int i)
{ this.i = i; }
@override
public String toString()
{ return "i value :"+i }
```

purpose of overriding

→ toString is non-static method of object class which will be inherited to every class of java

→ to string method will be available in every object of java.

→ to String method will return the string representation of the object
i.e it will give fully qualified classNameme of the object and hexadecimal converted number of that objects memory address in string format.

→ fully qualified ClassName means packageName.ClassName eg: packA @ 123F34

@ is also called as atlas because we cannot get the exact Memory Address in java.

→ Whenever we print the refrence variable in java there will be auto implicit call to string() method of that object

→ to String is a non-final method of object class and any subclass can override it

→ toString() method will be overridden to return that contents of the object in the String format.

→ toString() Method return type is String and its Complete Signature is

```
public String toString()
{
}
```

equals() Method

→ equals() is non-static method of object class which will be inherited to every class of Java.

→ equals() method will be available in every object of Java.

→ equals() method of object class will compare two objects equality based on memory address.

→ equals() is not a final method, and any subclass can override it.

→ equals() method will always be overridden to compare object equality based on the content of the objects (contents refers to attributes or variables)

→ equal() method return type is boolean and its Complete Signature is:

```
public boolean equals(Object obj)
{
}
```

Tester ②

```
main
{ A rv1 = new A();
  A rv2 = new A();
  Sop( rv1.equals(rv2)); false
  A rv3 = new A()
  A rv4 = rv3;
  Sop( rv3.equals (rv4)); true
}
```

A ①

Object

(1) B b1 = new B();

(2) Object obj = new B();

B b1 = (B) obj

to access the hidden feature of Obj

B

Tester ③

```
main ()
{ B rv1 = new B(10);
  B rv2 = new B(20);
  Sopln( rv1.equals (rv2)); false
  B rv3 = new B (40) ;
  B rv4 = new B(40);
  Sopln(rv3.equals(rv4)); true
```

B

```
int i;
B (int i)
{this.i = i;}
@override
public boolean equals (Object Obj)
{ B b1 = (B)Obj; // downcasting
  if (this.i == b1.i)
  { return true;}
  else return false
```

Interview

Q. What is the difference b/w comparision operator (==) and equals() method

→ There is no operator overloading or overridding but there is a concept of method overridding in java so, we ~~cannot~~ can change the implementation of equals() method but we cannot change the implementation of comparision operator (==)

Difference b/w java and C++

①	java is platform Independent	C++ is platform dependent
②	No operator overloading in java only method overloading	operator overloading in C++
③	Java doesnot support multiple Inheritance through classes but support upto some extent through Interface	C++ supports multiple Inheritance
④	In java there is only Constructor No destructor	C++ have both constructor and destructor

hashCode()

```
Tester
main()
{ A rv1 = new A();
  Sopln(rv1.hashCode());
  A rv2 = new A();
  Sopln(rv2.hashCode());
  A rv3 = new A();
  A rv21 = rv1;
  Sopln(rv3.hashCode());
  Sopln(rv21.hashCode());
```

```
A
Date
Page
```

0039797
27189673
354323
354323

overridding hashcode based on content

```
public class Tester
{ main
  { B rv1 = new B(10,20);
    Sopln(rv1.hashCode());
    B rv2 = new B(40,50)
    Sopln(rv2.hashCode());
  }
}
```

30
90

```
B
int i, j;
B(int i, int j)
{ this.i = i; this.j = j; }
@override
public int hashCode()
{ return i+j; }
```

→ hashCode() is a non static method of object class which will be inherited to every class of java

→ hashcode() method will be available in every Object in Java

→ hashcode() will return the unique integer Number based on the memory address of the object

Note:- ~~hashcode~~ The unique integer number is also called as hashnumber. It will be generated on hash algorithm

→ hashcode() method is not a final method of Object class and any subclass can override it

→ hashCode() method return type is int and its complete signature is

```
public int hashCode()
{                    }
```

→ hashCode() method will be overridden to return the hashnumber based on the content of the object

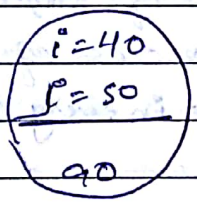
→ Whenever we need to compare two objects equality we should Override both equals() and hashCode() methods and procedure is as follows

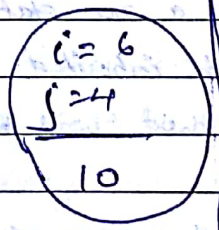
① call hashCode() method on the two objects and get the hashnumber

② Compare two hashnumber, and if both # are same then calls equals method

③ If hashnumber are different of the two object then no need to call equals method because if hashnumbers are different then object are not equal

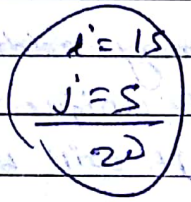
Note ~~Note~~ we should always call hashCode() method for comparision instead of equals() method because comparision through equals() method will be slow



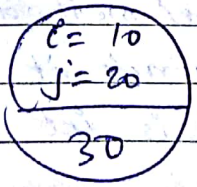


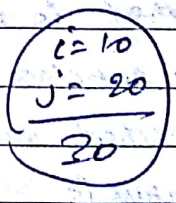
it is used in collections to avoid duplicates

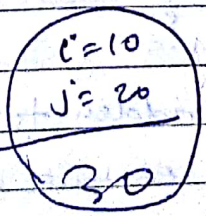
Specially set add method

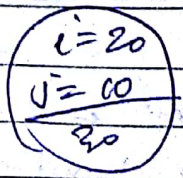


once hashCode() and equals() method is overrided properly duplicates can be avoided based on content









2. String is a final class ⇒ No subclass can be created to String

5. Implements Comparable Interface ⇒ we can sort String based on Content

Date
Page

String Class

→ String is a class

→ String is a built in class in java

1 → String class is available in java.lang package

→ String class is developed to handle string literals

→ There are 2 ways of creating string object

① Using new operator

② Using double quotes (" ")

→ String variables are reference variable

Note :- To use string class in any other package, there is no need to import java.lang.string

→ String is also a subclass to Object class

→ All the non-static methods of object class will be inherited to string class as well

*** → In string class, the three methods of the object class has been overrideen based on string value or String literals

→ toString() method of string class will return the string value of the object

→ equals() method of string class will return true if String objects are having same string value otherwise false (Case-sensitive)

→ hashcode() of string class will return (give) a unique hashnumber based on the String value

eg

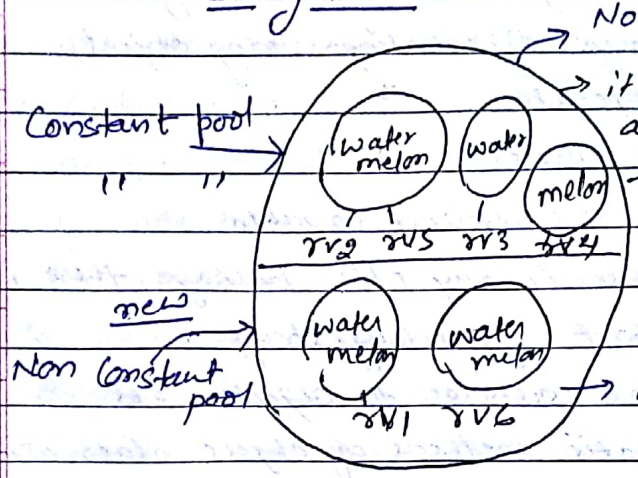
```
PSVM (
{ String rv1 = "Hey";
  String rv2 = "Hey";
  Sop(rv1);                    Hey
  Sop(rv2);                    Hey
  Sop("====");                 ====
  Sop(rv1.equals(rv2));        true
  Sop(rv1==rv1);               true
  Sop("====");                 ====
  Sop(rv1.hashcode);           72444
  Sop(rv2.hashcode);           72444
```

```
PSV Main (
{ String rv1 = new String("Hello");
  String rv2 = new String("Hello");
  Sop(rv1);                    Hello
  Sop(rv2);                    Hello
  Sop("======");               ======
  Sop(rv1.equals(rv2));        true
  Sop(rv1==rv2);               false
  Sop(rv1.hashcode);           69609680
  Sop(rv2.hashcode);           69609650
}
```

6. Implements Serializable => we can sesialise String object
7. String objects are immutable => we cannot change string object state once created
8. + operator overloaded only for string [any thing can be added to string]

→ Never compare two string using double equal operator (==)
Always use equals() method to compare string

String Pool




```
PSVM (String[] args)
{ String rv1 = new String("water melon");
  String rv2 = "water melon";
  String rv3 = "water";
  String rv4 = "melon";
  String rv5 = "water"+"melon";
  String rv6 = "water"+rv4;
  Sop(rv1==rv2);          false
  Sop(rv2==rv5);    →     true
  Sop(rv1==rv6);          false
  Sop(======);            ======
  Sop(rv1.equals(rv2));   true
  Sop(rv2.equals(rv5));   true
  Sopln(rv1.equals(rv6)); true
```

All are different when concatinate

```
s1 = "ja"
s2 = "va"
s3 = "java"
s4 = s1.concat(s2)
s5 = s1.concat("va")
s6 = "ja".concat(s2)
s7 = "jav".concat("va")
```

s3 = s4 = s5 = s6 = s7 → all false

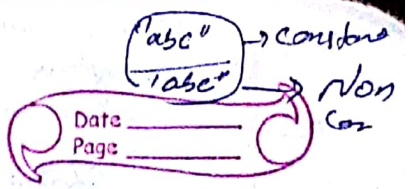
→ All the string object will be created in string pool
→ String pool is further divided into two pools
① Constant pool
② Non-Constant pool

```
String s1 = new String("abc");
Stirg s2 = new Stirg("abc");
```

→ Here 3 string object ("abc" ← s1, s2)

String s2 = new String("abc");

→ How many string object → 2
one "abc" in constant



→ String object created using double quotes will be stored in constant pool

→ String object created using new operator will be stored in Non-Constant pool

→ In Constant pool, duplicate string objects are not allowed

→ In Non-Constant pool, duplicate string objects are allowed

Note:- Duplicate string object means two string objects having same string values or literals

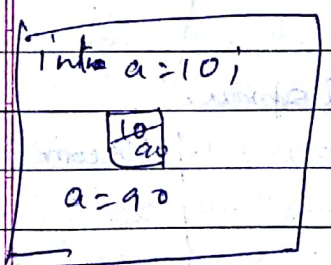
→ While concatenating two string if we are using a reference variable then the resultant string will be created using new operator.

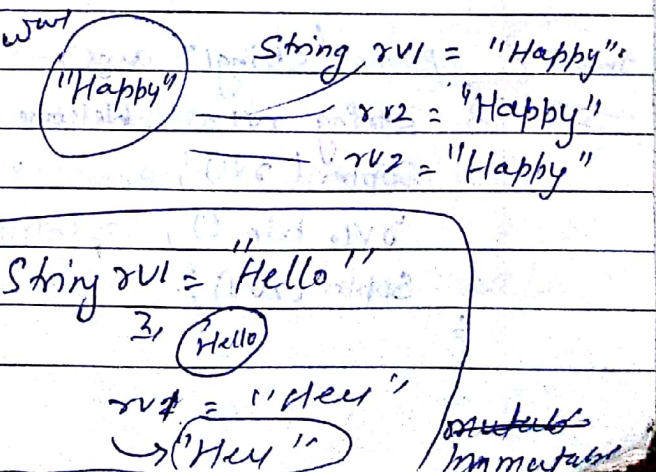
eg- String s1 = "Hi";
String s2 = s1 + "Hello";

→ Always we should compare string values using equals method

How String is immutable?

→ Whenever we create any string class object with some string value then we cannot ~~any~~ make any changes to that string class object.
We cannot make any changes to the existing string object's string value so we say string is class is immutable.





String s1 = null;
s1.length()

String s2 = NULL
s2.length()

String s3 = "null";
s3.length()

String function() approx 70

```
PSVM (String[] args)
{
String rv = "I LOVE INDIA";
```

* (1) `sop(rv.length());`	12	
* (2) `sop(rv.charAt(4));`	V	
(3) `sop(rv.indexOf('V'));` can be string argument also	4	
(4) `sop(rv.indexOf('I'));` -ve when char not available	0	
`sop(rv.lastIndexOf('I'));` eg "Love"	10	always 1st index
(5) `sop(rv.indexOf('I',3));`	7	
`sop("=========");`	=========	
(6) `sop(rv.toLowerCase());`	i love india	
(7) `sop(rv.toUpperCase());`	I LOVE INDIA	
(8) `sop(rv.startsWith("I LOVE"));`	true	
(9) `sop(rv.endsWith("MANDYA"));`	false	
`sop("======");`	======	
(10) `sop(rv.substring(2));`	LOVE INDIA	
11 `sop(rv.substring(2,9));` (2 inclusive, 9 exclusive)	LOVE IN	

```
}
```

+ operator overloading of String

```
String s1 = "abc";
s1 = s1+123;
s1 = s1+true;
s1 = s1+null;
s1 = sop.45+s1;
```

All interview *

```
s1 = "test";
s2 = 3+4+s1;   7test
s3 = 4+s1+3;   4test3
s4 = s1+4+3;   test43
String s1 = 4+5 → CTE
s1 = null;
s2 = s1+9;     null9
s5 = 9+s1;     9null
s4 = s2+s3;    null99null
s5 = null+s1;  nullnull
s6 = s1+null;  nullnull
String s1 = null+null → CTE
```

(14)

```
PSVM (String[] args)
{ String rv = "  Mubai  ";
12  sop(rv.trim());
    String rv1 = "  Mubai is in India  ";
    sop(rv1.trim());
13  sop(rv2.contains("Mumbai"));
}
```

Mumbai

Mumbai is in India

true

Interview Q

```
PSVM (String[] args)
{ String rv1 = "  Welcome to Q Spiders.  ";
  Sopln(rv1);
  rv1.trim();
  Sopln(rv1);
}
```

Welcome to Q Spiders.

Welcome to Q Spiders

char x = {'h','e','l'
,'l','o'};
String s1 = new String(x);

String s1 = "hello";
char[] x = s1.toCharArray();

Reversing a String

```
PSVM(String[] args)
{ String rv = "SREDIPSG";
  Sopln("The original string :: "+rv)
  Sopln("The reversed string :: "+rv)
  for(int i = rv.length()-1; i>=0; i++)
  { sop(rv.charAt(i);}
}
```

Original string :: SREDIPSG
Reversed string :: GSPIDER

All interview *

Java program to reverse a string & check Palindrom

```
PSVM(String[] args)
{ String rv1 = "DAD";
// rv1 = rv1.toUpperCase();
  String rv2 = "";
  for(int i = rv1.length()-1; i>=0; i--)
  { rv2 = rv2 + rv1.charAt(i);}
  Sop("The original string :: "+rv1);
  Sopln("reversed string :: "+rv2);
  if (rv1.equalsIgnoreCase(rv2))
  { sop("So, It's a Palindrome");}
  else { Sopln("So, Its NOT a Palindrome");}
}
```

```
String s1 = "hello"
char x = s1.toCharArray();
for(int i=0; i<x.length/2; i++)
{ char temp = x[i];
  x[i] = x[x.length-1-i]
  x[x.length-1-i] = temp;
String s2 = new String(x);
```

Original String :: DAD
Reversed string :: DAD

So, Its Palindrome

reverse string use Recursion

Input string rv = "abc"
Output string = "aa bb cc"

```
PSVM(String[] args)
{ String rv1 = "abc";
  String rv2 = "";
  for(int i=0; i<rv1.length; i++)
  { char ch = rv.charAt(i);
    rv2 = rv2 + charAt(i) + charAt(i);}
  Sopln("Given string = "+rv1);
  Sopln("Result String = "+rv2)
}
```

```
revString(String s)
if(s.length()==0)
  return "";
return
s.charAt(s.length()-1)
+ rev(s.substring
(0, s.length-1))
```

abc
aabbcc

split ( ) → accepts Regex Unlike StringTokenizer

[a-g] [a-g][1-5]

\\d → anydigit → reverse → \\D
\\s → whitespace → rev → \\S
\\w → word char → \\W

Count the no of occurance of a in malayalam

```
PSVM(String[] args)
{ String rv = "malayalam";
  int count = 0;
  for(int i=0; i<rv.length(); i++)
  { char ch = rv.charAt(i);
    if (ch == "a");
    { count++; }
  }
  Sopln("count = " + count);
```

4

1 input string = QSPIDERS result string = QSIDERS

```
PSVM(String[] args)
{ remove("QSPIDERS", 0);
  remove("QSPIDERS", 1);
  remove("QSPIDERS", 2);
}
Static void remove(String rv1, int n)
{ String s1 = rv1.substring(0,n);
  String s2 = rv1.substring(n+1)
  String s3 = s1+s3
  Sopln(s3);
}
```

SPIDERS
QPIDERS
QSIDERS

Q. Comparing 2 Strings

```
PSVM(String[] args)
{ String rv1 = "Hello India How are you";
  rv2 = "Hello world";
  rv3 = "Hey world, How are you";
  rv4 = "Hey"
  rv5 = "Hey"
  rv6 = "hey"
  Sop(rv1.codePointAt(0));
  Sop(rv2.compareTo(rv3));
  Sop(rv2.compareTo(rv3));
  Sop(rv4.compareTo(rv5));
  Sop(rv5.compareTo(rv6));
  Sop(rv5.compareToIgnoreCase(rv6));
}
```

Unicode value	72
+ve No. if 1st > 2nd	21
-ve No. if 1st < 2nd	-13
0 if 1st == 2nd	0
Uses unicode value of H&h	-32
	0