Chapter - 1 : Strings

Revision Notes

In java, string is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

char[] ch={'j','a','v','a','t','p','o','i','n','t'};

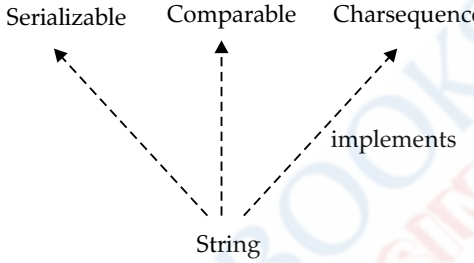
String s=new String(ch);

is same as:

String s="javatpoint";

Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, Comparable and *CharSequence* interfaces.



- **How to create String object?**

  There are two ways to create String object:

  By string literal

  By new keyword

  (1) String Literal

  Java String literal is created by using double quotes. For Example:

  String s="welcome";

  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

  (2) By new keyword

  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).

  Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

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 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, Char Sequence... 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 object
11.	boolean is Empty()	checks if string is empty
12.	String concat(String str)	concatenates specified string
13.	String replace(char old, char new)	replaces all occurrences of specified char value
14.	String replace(CharSequence old, CharSequence new)	replaces all occurrences of specified CharSequence
15.	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
16.	String[] split(String regex)	returns splitted string matching regex
17.	String[] split(String regex, int limit)	returns splitted string matching regex and limit
18.	String intern()	returns interned string
19.	int indexOf(int ch)	returns specified char value index
20.	int indexOf(int ch, int fromIndex)	returns specified char value index starting with given index
21.	int indexOf(String substring)	returns specified substring index
22.	int indexOf(String substring, int from Index)	returns specified substring index starting with given index
23.	String toLowerCase()	returns string in lowercase.
24.	String toLowerCase(Locale l)	returns string in lowercase using specified locale.
25.	String toUpperCase()	returns string in uppercase.
26.	String toUpperCase(Locale l)	returns 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 overloaded.

Know the Terms

- String defines a sequence of characters. The String type is used to declare string variables. You can also declare arrays of strings. A quoted string constant can be assigned to a String variable. A variable of type String can be assigned to another variable of type String. You can use an object of type String as an argument to println( ).
- Java provides three classes as

  Character class can hold only single character.

  String class can hold strings that cannot be changed.

  String Buffer class can hold strings that can be changed or modified.
- Substring( ), string concatenation as concat( ) and string length as length( ) etc. are the operations defined on string.

❑❑

Chapter - 2 : Recursion

Revision Notes

- Recursion is a process by which a function or a method calls itself again and again. This function that is called again and again either directly or indirectly is called the "recursive function".

- How Can We Stop Infinite Conditions of Recursion in Java?
  To stop the infinite conditions, we must have the following:
  - **Base Condition :** Specify inside if the condition were to stop the recursion function.
  - **Recursion Call :** Proper recursion call.
- Call a recursion function
  We can call a recursion function in 2 ways
  1. Direct Recursion Call
  When we call the same method from the inside method body.
  Syntax :

```
returntypemethodName()
{
//logic for application
methodName();//recursive call
}
```

  2. Indirect/Mutual Recursion Call
  When we call a method from another method and another method called from the first method, vice versa.
  Syntax :

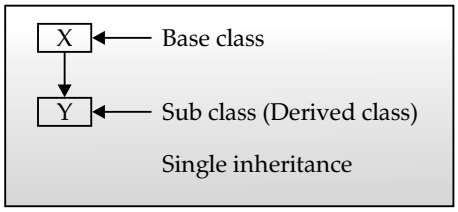
```
firstIndirectRecursive()
{
// Logic
secondIndirectRecursive();
}
secondIndirectRecursive()
{
//Logic
firstIndirectRecursive();
}
```

❑❑

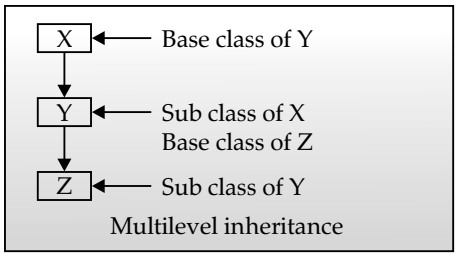
Chapter - 3 : Inheritance, Interfaces and Polymorphism

Revision Notes

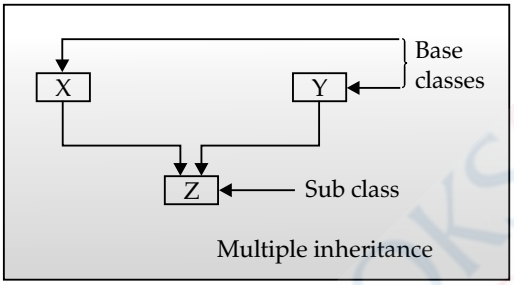
- Method overloading is one of the ways through which java supports polymorphism. Method overloading can be done by changing number of arguments or by changing the data type of arguments. If two or more method have same name and same parameter list **but differs in return type are not** said to be overloaded method
- Forms of inheritance
- **Single inheritance :** When a subclass inherits from only one base class, and there are only 2 classes in this relationship, then it is known as single inheritance.



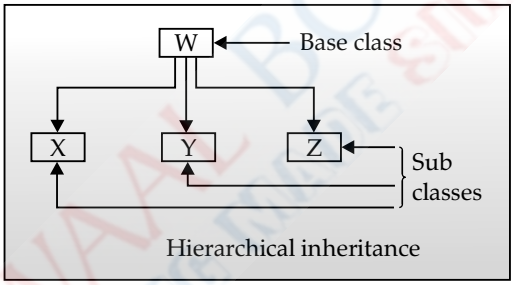
- **Multilevel inheritance :** When a subclass inherits from a class that itself inherits from another class, it is known as multilevel inheritance.



- **Multiple inheritance :** When a subclass inherits from multiple base classes it is known as multiple inheritance. Java does not support multiple inheritance.



- **Hierarchical inheritance :** When many subclasses inherit from a single base class it is known as hierarchical inheritance.



Know the Terms

- **Extends** is the keyword used to inherit the properties of a class
- The **implements** keyword is used with classes to inherit the properties of an interface.
- **This** keyword is used to refer to current object.
- Super keyword is used to refer to immediate parent class of a class.
- Overriding - to override the functionality of an existing method.

❑❑

Chapter - 4 : Data Structures

TOPIC-1
Queues and Stacks

Revision Notes

- Public class Stack<Item>implements Iterable<Item>

Stack ()	create an empty stack
boolean is Empty ()	is the stack empty ?
void push (Item item)	push an item onto the stack
Item pop ()	return and remove the item that was inserted most recently
int size ()	number of items on stack

- Public class Queue<Item>implements Iterable<Item>

Queue ()	create an empty queue
boolean isEmpty ()	is the queue empty ?
void enqueue (Item item)	insert an item onto queue
Item dequeue ()	return and remove the item that was inserted least recently
int size ()	number of items on queue

Know the Terms

- **Enqueue (qstore) :** adding an element to the queue at its rear
- **Dequeue (qstore) :** deleting an element at the front of a queue.
- **Empty :** status of queue being full.
- **Full :** status of a queue being full.
- **Front :** Refers to the first element of the queue.
- **Rear :** Refers to the last element of the queue.

TOPIC-2

Trees

Revision Notes

- A binary tree is made of nodes, where each node contains a "left" pointer, a "right" pointer, and a data element.
- The "root" pointer points to the topmost node in the tree.
- The left and right pointers recursively point to smaller "subtrees" on either side.
- A null pointer represents a binary tree with no elements -- the empty tree.
- The formal recursive definition is : a **binary tree** is either empty (represented by a null pointer), or is made of single node, where the left and right pointers (recursive definition ahead) each point to a **binary tree**.
- A "binary search tree" (BST) or "ordered binary tree" is a type of binary tree where the nodes are arranged in order : for each node, all elements in its left subtree are less-or-equal to the node (<=), and all the elements in its right subtree are greater than the node (>).

❑❑