CORE JAVA

(SUMMARY DOC)

INDEX

DEFINITION/CONCEPT	EXAMPLE	SYNTAX
JVM,JRE,JDK		
➢ Jvm-It is an abstract machine which provides runtime environment in which java byte code can be executed. ➢ Performs 4 operations:- load code,verify code,execute code,provide runtime environment.		
➢ Jre-Java runtime environment ➢ It is the implementation of jvm that physically exists. ➢ Contains set of libraries plus files which the jvm uses at runtime.		
Jdk=Jre+development tools		
OOPS CONCEPTS		
Object is a real world entity such as a pen,chair etc. It simplifies software development and maintenance by providing some concepts such as:- ➢ Objects ➢ Classes ➢ Inheritance ➢ Polymorphism ➢ Abstraction ➢ Encapsulation		
Naming Conventions:- ➢ Classes-Start with capital letter and should be a noun. ➢ Interfaces-Start with a	Eg.System,String	

capital letter and should be an adjective ➢ Packages-small letter ➢ Variables-should begin with lowercase ➢ Constants-Should be in upper case ➢ Methods-should begin with lower case and be a verb.	Eg.Runnable,ActionListener Eg.java,util,sql Eg.firstname Eg.RED,YELLOW Eg.main(),print(),println()	
Objects and Classes:-		

Object:
➢ Objec t is a real world entity which has:
1.)State-represents the data of an object 2..)Behavior-represents the behavior of an object.
3.)Identity-implemented via a unique identity.The value of id is not visible to the external user.it is used by the jvm.
➢ Object is an Instance of a class.
	sampleclass s=new sampleclass();
	s.insertrecord(a,b);
	◆ above is object creation using new.
	◆ the values a and b are stored in the heap.
	◆ The variable s is the pointer to them in the stack.	

Class:
➢ A class is like a blueprint or template from which an object can be created.
➢ A class can contain:
1.)data member
2.)methods
3.)constructor
4.)block
➢ A variable created inside class but used outside is instance variable.
	public class sampleclass
	{
	public static void main(String[] args)
	{
	sampleclass s=new sampleclass();
	String a="rohan";
	int b=10;
	s.insertrecord(a,b);
	}
	
	private void insertrecord(String a, int b)
	{
	System.out.println(a+" "+b);
	}
	
	}	

Way of creating object in java		

are: ➢ By new keyword ➢ By newInstance method ➢ By clone() method ➢ By factory method etc.		
ANONYMOUS OBJECTS An object which has no reference.	new Anonymousobject().somemethod(); ◆ Anonymousobject is the class name. ◆ Somemethod is the method name.	
METHOD OVERLAODING Same as C++.		
CONSTRUCTOR BEAHVIOR ➢ Constructors can be overloaded. ➢ They can be automatically generated by the ide. ➢ Copy constructor not present. ➢ So to copy value of one object to another,we can use a constructor,assign value of one object to another,by clone method of object class.		
Static Keyword: Static keyword is used mainly for memory management. It can be used with ➢ variables ➢ methods ➢ separate block.		
Static Variable Can be used to common property of all objects. Memory assigned when class is loaded.	int rollno; String name; static String col="ASE"; staticusage s=new	

	staticusage(1,"rohan"); ➢ The above line is used to create a object of the class student. ➢ The rollno and names change for a student. But college remain same. ➢ So by making it static we are conserving memory.	
Static Method ➢ A static method belongs to a class than the object of a class. ➢ Hence to access the method we need not create an object of the class. ➢ Can access static variable and can change its values. ➢ Restrictions-cannot use non-static variables and this,super cant be used inside.	Math class. WE call it using Classname.function() i.e Math.random()	

Static block:
Used to initialize static data members.
Executed before main when class is being loaded.
Shouldnt raise an exception.
	public class staticmethndblock
	{
	
	static {
	System.out.println("will be executed before main");
	}
	public static void main(String[] args)
	{
	System.out.println("main starts");
	somemethod();
	}
	
	private static void somemethod()
	{
	System.out.println("object not needed to call this method");
	}
	}
		Output is:
		
		will be executed before main
		main starts
		object not needed to call this method

ARRAYS:
➢ Array is a collection of similar types of elements
	public static void main(String[] args)
	{
	int[] a=new int[3];
		Data_type[] name=new Data_Type[size];

having a contiguous memory location.
➢ Optimizes code and can be accesses at any place as it is index based.
➢ Types of arrays:
1.)Single dimensional
2.)Multi dimensional
➢ Copying arrays done by System.arraycopy method.
	int[][] b=new int[2][2];
	char[] copyfrom={'a','b','c','d','e'};
	char[] copyto=new char[6];
	
	System.arraycopy(copyfrom,0,copyto,1,4);
	System.out.println(new String(copyto));
	
	}
		arraycopy-
		
		First parameter-source
		Second parameter-source position
		Third Position-Destination
		Fourth position-destination position
		Fifth position-length

Call By Value ➢ There is only call by value in java. ➢ If changes to value are done in the called method,the value is not affected in the calling method.		

Command Line Arguments
➢ These are argumensts that are passed at the time of running of the program.
➢ These arguments can be received by program and used as input.
➢ To pass arguments In run-->Edit configurations-->pass parameters.
➢ Parameters needs to be necessarily be passed for the adjacent code.
	public static void main(String[] args)
	{
	System.out.println("first argument is"+args[0]);
	}	

Access Modifiers- There are two types of modifiers. They are:- ➢ Access modifiers-specifies the accessibility of a method,data member or a constructor.eg.public,private ➢ Non access modifiers-		

eg.Abstract,synchronized ,static.		

Access Modifiers:
The access modifiers in java are:-
➢ **Private**-A private access modifier is accessible only inside class. Classes except for nested cant be private or protected.
➢ **Default**-Accesible only within package. If no access modifier is declared this is the default.
➢ **Protected**-Accesible within package.For accesiing from outside the class needs to be a the class declared as protected.The methods of this class cant access the functions declared as protected.That is class outside the package has to extend the class containing protected method.
➢ **Public**- Accessible everywhere. Widest scope
➢ The scope of the overridden functions can only be increased. eg.if class A extends B and B has a function F1() which is default,then in B it can only be made protected or public.
	public static void main(String[] args)
	{
	f1();
	f2();
	f3();
	f4();
	
	}
	
	protected static void f4()
	{
	System.out.println("access modifier protected");
	}
	
	static void f3()
	{
	System.out.println("access modifier default");
	}
	
	public static void f2()
	{
	System.out.println("access modifier public");
	}
	
	private static void f1()
	{
	System.out.println("access modifier private");
	}
		f2 is accessible outside package and everywhere in package.
		f4 accessible inside package after the class outside package extends it.
		f1() accessible only in the class
		f2()-only inside package

This keyword It is a reference variable which refers to the current object. Resolves ambiguity between instance variable and parameter. This has a variety of uses:-		

➢ To refer class instance variables. ➢ To invoke Current class constructor. ➢ To invoke current class method. ➢ Passed as an argument to in method . ➢ Passed as an argument in constructor . ➢ To return current class instance.		

To refer class instance variables
 	
➢ this.id is the instance variable and id is the local variable
	public InstanceVarRefer(int id, String name)
	{
	this.id = id;
	this.name = name;
	}
	
	int id;
	String name;
	
	public static void main(String[] args)
	{
	InstanceVarRefer a=new InstanceVarRefer(1,"a");
	System.out.println("id= "+a.id+" "+"name= "+a.name);
	}	

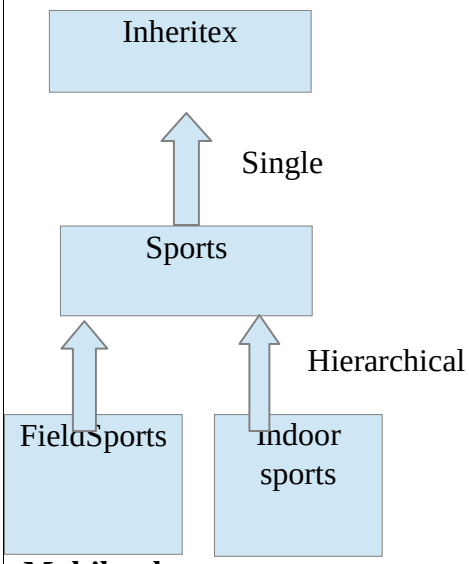
To invoke Current class constructor
 	
➢ The call to the constructor should be made in the very first line of the constructor
➢ Use to reuse constructor in constructor (constructor chaining)
	public InstanceVarRefer(int id, String name)
	{
	this();
	this.id = id;
	this.name = name;
	
	}
	
	public InstanceVarRefer()
	{
	System.out.println("this is an empty constructor");
	}	

To invoke current class method
 	
➢ Compiler automatically uses this by default for method calls.
	public static void main(String[] args)
	{
	
	someefunc2();
	
	}
	
	private static void someefunc2()
	{
	System.out.println("testing this");
	}
	
	private void somefunc()
	{
	this.someefunc2();
	
	}	

Passed as an argument in method
 	
➢ Mainly used for event handling. i.e reference of one class has to be provided to other.
	private void m()
	{
	p(this);
	}
	private void p(Object o)
	{
	System.out.println("this is passes in method");
	}	

Passed as an argument in constructor ➢ Useful when we have to use one object in multiple classes.		

To return current class instance variable
 	
➢ Return type must be class type**.**
	private InstanceVarRefer return_this()
	{
	return this;
	}
		Return_type name()
		{
		return this;
		}

Inheritance
➢ Mechanism through which one object of a derived class acquires the properties and behavior of an object of a super class.
➢ Used for method overriding and code reusability.
➢ Types of inheritance in Java are single,multilevel,hierarchical.
➢ The extends keyword is used by subclass to inherit from superclass.
➢ Is A relationship
	
	Multilevel
	Inheritancex->sports-->Firld sports
		Class Sub_class extends Super_Class
		{
		
		}

Aggregation
➢ if a class has entity reference,it is said to be aggregation.
➢ It is HAS-A relationship
➢ used for code re usability
➢ Inheritance should be used only if relationship is-a is maintained during lifetime of object.
➢ Else use aggregation.
➢ To string has to be overidden
➢ using inheritance here would have been a bad option here.
	public class Student
	{
	int id;
	String name;
	Address addr;
	public static void main(String[] args)
	{
	Address addr=new Address(1,"muni garden","bangalore","karnataka",5600);
	Student s=new Student(1,"rohan",addr);
	System.out.println(s.id+" "+s.name+" "+s.addr);
	}
	}
	Address class contains:
	◆ doorno
	◆ street name
	◆ city
	◆ state
	◆ pincode	

Method overriding ➢ having a base class method in subclass with same signature(same name and parameters(is		

method overriding. ➢ Used to provide specific implementation for a method which already provided by superclass ➢ For run time polymorphism ➢ Static methods cant be overridden.		

Runtime polymorphism
➢ A mechanism through which call to an overridden method is resolved at runtime and not compile time.
➢ The determination of method called is based on the the object referenced by the reference variable.
➢ Only methods can participate in rtp not data members.
	public class RtpBike
	{
	int a=50;
	void run()
	{
	System.out.println("bike is running");
	}
	}
	
	public class Honda extends RtpBike
	{
	int a=100;
	void run()
	{
	System.out.println("honda bike is running");
	}
	
	}
		public class RtpMain
		{
		public static void main(String[] args)
		{
		RtpBike r=new Honda();
		r.run();
		System.out.println(r.a);
		}
		}
		
		Output:
		honda bike is running
		50

Final
➢ Used to restrict the user.
➢ Can be used with variables,methods and classes.
	public class FinalTEst
	{
	final int i=50;
	final void func1()
	{
	System.out.println("Final method func1 cant be overidden");
	}
	void testvar()
	{
	System.out.println("value of final variable\t"+i+"cannot be changed ");
	//i=100;
	}
	}
	
	public final class Finalmeth extends	

	FinalTEst
	{
	void finalclassmethdo()
	{
	System.out.println("this class cant be extended");
	}
	}	

➢ **Final variable**-A variable whose value is declared final cant be changed.
➢ **Final method-**a method which is declared final cant be overridden.
➢ **Final class**-a class declared final cant be extended.
	public class Testmain
	{
	public static void main(String[] args) {
	FinalTEst f=new Finalmeth();
	f.func1();
	f.testvar();
	
	}
	}
	Output:
	Final method func1 cant be overridden
	value of final variable50cannot be changed	

Super-Super is a reference variable that is used to refer to immediate parent class object. It can be used to:-
➢ Immediate parent class reference instance variable.-his is done by calling super.variable name from derived class.
➢ Immediate parent class constructor-super() should be called in first line of derived class constructor
➢ Immediate parent class method. -his is done by calling super.methodname from derived class.
	public final class Finalmeth extends FinalTEst
	{
	public Finalmeth()
	{
	super();
	}
	
	void finalclassmethdo()
	{
	
	System.out.println("this class cant be extended");
	System.out.println("called through super in sub class"+super.i);
	super.func1(); //using super to call a method
	}
	
	}
	public class Testmain
	{
	public static void main(String[] args) {	

	FinalTEst f=new Finalmeth();
	f.func1();
	f.testvar();
	
	}
	}	

REFLECTION ➢ Process of modifying or examining the behavior of a class at runtime. ➢ The java.lang.Class class provides methods to get metadata ,examine and change the runtime behavior of the class. ➢ Reflection api mainly used in ide,debuggers and testing tools.		
Java.lang.Class class performs mainly two tasks:- ➢ Provides methods to get metadata of class at runtime. ➢ Provides methods to examine and change the runtime behavior of a class .		
Getting object of a Class class The ways are:- ➢ forName() method of Class class. ➢ Getclass() method of object class. ➢ The .class syntax		

forName() method of Class class
 	
➢ Used to load the class dynamically.
➢ Returns the instance of the Class class.
➢ Should be used if we
	public static void main(String[] args)
	{
	simpleclass s=new simpleclass();
	findcla(s);
	Class cs=String.class;
	System.out.println("using .class is "+cs);
	Class c= null;	

know the fully qualified name of a class.
➢ Cant be used on primitives.
	try {
	c = Class.forName("reflectionprac.simpleclass");
	} catch (ClassNotFoundException e) {
	e.printStackTrace();
	}
	System.out.println("using getname is "+c.getName());
	}
	
	private static void findcla(Object s)
	{
	System.out.println("using get class is "+s.getClass());
	
	}	

Getclass() method of object class ➢ It returns the instance of a Class class. ➢ Should be used if object of class is present.	Output:- ✓ using get class is class reflectionprac.simpleclass ✓ using .class is class java.lang.String ✓ using getname is reflectionprac.simpleclass	
The .class syntax ➢ If a type is available but no instance of it is available,then it is possible to obtain a class by appending a .class to name of the type. ➢ Can be used for primitive types also		

NewInstance Method
➢ Creates a new instance of the class.
➢ The folowing methods can eb used to determine type of class object.
➢ Isarray(),isInterface(), isprimitive()
➢ All of the above returns a boolean value
	c=Class.forName("reflectionprac.classobj.classobj.classobj.simpleclass");
	simpleclass x=(simpleclass)c.newInstance();
	x.printmesg();
	if(!c.isArray())
	{
	System.out.println("class is not an array");
	}
	else if(c.isInterface())	

	{
	System.out.println("object is of an interface");
	}
	else if(c.isPrimitive())
	{
	System.out.println("object is a primitive");
	}	

Javap util:-		

Accesing the private behavior of a class
➢ done by by changing its runtime behavior.
	Class c=Class.forName("reflectionprac.classobj.classobj.aceesprivate.a");
	Object o=c.newInstance();
	Method m= c.getDeclaredMethod("pri",null);
	m.setAccessible(true);
	m.invoke(o,null);	

Resource Bundle:-
➢ A file which can be used to store important configuration details so that when there is a change in a project configuration can be done and not coding.
➢ Reduces recompiling the whole code.
➢ The file should have a suffix .properties. eg hello.properties.
➢ The class is decided at runtime.
➢ Other configuration details also can be included.
	public static void main(String[] args) {
	String cn= ResourceBundle.
	getBundle("resourcebundle").
	getString("ar");
	String dn= ResourceBundle.
	getBundle("resourcebundle").
	getString("li");
	
	Class c=Class.forName(cn);
	bject o=c.newInstance();
	System.out.println(o.getClass());
	
	Class c1=Class.forName(dn);
	Object o1=c1.newInstance();
	System.out.println(o1.getClass());
	
	}
	}
		Resource file:
		
		ar=JavaOops.arrayprac.Arrayconc
		st=JavaOops.staticusage.staticvar
		li=collections.prac.list.listconcepts

Interfaces ➢ An interface tells what a class can do but not how it does it. ➢ An interface can extend other interfaces. ➢ Designed to support		

dynamic execution at runtime. ➢ An interface variables are static and final. ➢ For a class to implement an interface,a Implements keyword has to be used. ➢ A class which partially implements an interface is needs to be declared as an abstract class ➢ Implementations can be accesses through interface reference.	Interface a=new ClassImplementinginginterface();	
RTTI		

Abstract classes
An abstract class is one which has normal methods and abstract methods.
 	
It cant be instantiated.
 	
Used to achieve abstraction.
 	
The abstract methods has to be overidden my extending class.
 	
The other methods can be directly called..
	public class Honda extends Bike
	{
	
	@Override
	void win()
	{
	System.out.println("winning");
	}
	
	public static void main(String[] args) {
	Bike b=new Honda();
	b.run();
	b.win();
	}
	}
	ABSTRACT CLASS
	
	abstract class Bike
	{
	void run()
	{
	System.out.println("running");
	}
	abstract void win();
	}	

RMI		
Instance of:	public class Honda extends Bike {	

➢ It is used to check whether an object is an instance of a specific type(class or subclass or interface or not)
➢ Downcasting can be done using instance of.
	@Override
	void win()
	{
	System.out.println("winning");
	}
	
	public static void main(String[] args)
	{
	Bike b=new Honda();
	
	System.out.println(b instanceof Bike);
	System.out.println(b instanceof Honda);
	if(b instanceof Bike)//downcasting
	{
	Honda h=(Honda)b;
	}
	
	// b.run();
	//b.win();
	}
	}	

EXCEPTIONS ➢ Provides mechanism to handle runtime exceptions so that normal flow of code can be maintained. ➢ Only functions throw exceptions not classes. ➢ An error means program cannot be recovered. ➢ An exception means program can be recovered. ➢ Both exception and error classes extends Throwable. ➢ Control shifts from try to catch block when an object of the exception is thrown.		

STEPS FOR AN EXCEPTION
 	
➢ System should know something is an exception
➢ System should know when it occurs.an object of exception is thrown from try to catch block
➢ Remedial action should be known.
	Step 1:class e1 extends Exception{}
	
	Step 2:
	private static void f1() throws e1 {
	int ran= (int) (Math.random()*100);
	if(ran%3==0) //
	{
	throw new e1();
	}
	else
	{
	System.out.println("no exception thrown");
	}
	}
	Step 3:
	try {
	f1();
	} catch (Exceptions.practice.e1 e1)
	{
	e1.printt();
	e1.printStackTrace();
	}
	}
	}	

Exception Spawning or chaining:
➢ Whenever in a program the first exception causes an another exception, that is termed as Chained Exception.
➢ Java provides new functionality for chaining exceptions.
	public class ExceptionSpawning
	{
	public static void main(String[] args)
	{
	try
	{
	f1();
	}
	catch (ArithmeticException e)
	{
	throw new IllegalArgumentException("exception chain",e);
	}
	
	}
	private static void f1() throws ArithmeticException
	{
	int i;	

	i=50/0;
	}
	
	}	

FINALLY BLOCK
 	
➢ It is a block which is always executed
➢ It is used to perform important task such as closing connection,stream etc.
➢ Mainly used to put clean up code
➢ Though here only a s.o.p line is written in finally it does above work.
	finally
	{
	System.out.println("finally is executed always");
	}	

THROW KEYWORD:
 	
It is used to explicitily throw an exception.
Checked or unchecked exceptions can be thrown.
Mainly used to throw custom exceptions.
	if(ran%3==0) //
	{
	throw new e1();
	}
	
	e1 is a custom exception	

THROWS KEYWORD:
Throws keyword is used to declare an exception.
It tells the programmer that an exception may occur in a function.
	private static void f1() throws e1
	{
	
	}
	
	}
		Void method_name throws exception_class_name()
		{
		
		}

Throw vs Throws: Throw: Used to throw an exception; Checked exceptions can be propogated without throw. Throws: Used to declare an exception; Checked exceptions can be propogated only with throws. Throw: Followed by instance of a Throws: Followed by a class.		

<table><tr><td>class.</td><td></td></tr><tr><td>Is used within the method.</td><td>Is used with method signature</td></tr><tr><td>We cannot throw multiple exceptions</td><td>Throws can be used with multiple exceptions</td></tr></table>			
CHECKED EXCEPTIONS ➢ Checked exception or Nonruntime exceptions are those that are detected by compiler before the compilation of our program. ➢ Checked Exceptions extend Class Exception ➢ Exception handling(try/catch,finally or throws) needs to be explicitily done by the user.	Class nonruntimeex1 extends Exception{} class runtimeex1 extends RuntimeException{} class runtimeex2 extends runtimeex1{} class nonruntimeex2 extends nonruntimeex1{}		

UNCHECKED EXCEPTIONS
➢ Unchecked exception or runtime exceptions are those that are not detected by compiler before the compilation of our program.
➢ Checked Exceptions extend Class RuntimeException.
➢ Exception is known only at runtime.
This is a problematic but it is preferred to checked exceptions as it ensures loose coupling.
	public class ChkdNdUnchkd
	{
	public static void main(String[] args)
	{
	
	firstrunex();
	
	try {
	firstnonrunex();
	} catch (Exceptions.practice.nonruntimeex1 nonruntimeex1) {
	
	nonruntimeex1.printStackTrace();
	}		

	}
	private static void firstnonrunex()
	throws nonruntimeex1
	{
	secondnonrunex();
	
	}
	private static void
	secondnonrunex() throws
	nonruntimeex1
	{
	throw new nonruntimeex1();
	}
	private static void firstrunex()
	{
	secondex();
	}
	
	private static void secondex()
	throws runtimeex1
	{
	throw new runtimeex1();
	}	

EXCEPTIONS DURING OVERRIDING
➢ A method which overrides another method which throws a checked exception cannot throw a new checked exception.
➢ It can throw a new unchecked or runtime exception.
	private static void secondnonrunex()
	throws nonruntimeex1 {
	throw new nonruntimeex1();
	}
	private static void
	secondnonrunrex(String x)
	{
	
	throw new runtimeex1();
	
	}
	
	➢ Here method secondnonrunex() throws a checked exception.
	➢ The overridden method can throw only an unchecked or runtime exception.	

COLLECTIONS		
LIST ➢ List is a one column	List l=new Arraylist();	

structure that can take duplicates in java. ➢ List is an interface ➢ Arraylist,vector and list are implementations of list. ➢ Though objects of different type can be added to a list,generally it makes sense to add objects of same type.	● correct way of creating an arraylist as we don't want to access the exclusive methods of arraylist.	

Add an Element
 	
➢ Use inbuilt method list.add to add an entry.
➢ User defined objects as well as predefined data types can be added.
➢ For user defined object to be read the Tostring() method has to be overidden.
	List l=new ArrayList();
	
	private static void addentry(List l)
	{
	user u=new user(1,"rohan");
	//add item
	l.add("hello");
	l.add(3) ;
	l.add(u);
	}	

ViewAll
 	
To use all elements in a list an iterator or for loop can be used. We have traversed through an iterator.
	private static void Viewall(List l)
	{
	Iterator itr=l.iterator();
	while(itr.hasNext())
	{
	System.out.println(itr.next());
	}
	}	

Viewparticular
 	
➢ Used to view a particular element in the list.
➢ Indexof method is used.
➢ And depending on the objects passed the position of the object in the list is returned.
	private static void viewparticular(List l)
	{
	String s="hello";
	Object obj=new user(1,"rohan");
	
	int pos=l.indexOf(s);
	if(pos!=-1)
	{
	System.out.println("element fount at "+pos);
	}	

	else
	{
	System.out.println("element not found");
	}
	}	

Modifying a list ➢ While modifying a list the concept of mutable and immutable objects comes into focus. ➢ Predefined datatype objects are immutable and they cant be modified. ➢ The object has to be removed and a new entry added. ➢ User defined objects can be modified.		
Sorting of list: Depending on the object two types of sort:- Natural sort -uses Collections class method Runtime sort_also uses collections class method.But requires a comparator object.		

Natural Sort
 	
Done for object of the same type.
Here we are sorting integres.
Likewise strings and other data types can be sorted.
Uses comparable interface.
	private static void addnormalelements(List l)
	{
	l.add(1578) ;
	l.add(5);
	l.add(2);
	l.add(57);
	Collections.sort(l);
	System.out.println(l);
	}	

Runtime sort:- Done for user defined object or sort based on a specific criteria.		

Steps
 	
➢ The user defined class has to implement comparable.
➢ Create a class which implements comparator.
➢ Overide the compareTo method based on the need.
➢ Create a new comparator object and pass that to sort method along with list.
	Class Implementing Comparator
	
	public class UBS implements Comparator<user>
	{
	
	@Override
	public int compare(user o1, user o2)
	{
	System.out.println("sorting based on userid");
	
	int i= o1.getId() ;
	int j=o2.getId();
	return o1.compareTo(o2);
	}
	}
		Method to sort()
		
		private static void adduserdefinedelements(List l)
		{
		user u1=new user(5,"a");
		user u2=new user(2,"b");
		user u3=new user(3,"c");
		l.add(u1);
		l.add(u2);
		l.add(u3);
		Comparator<user> es=new UBS();
		Collections.sort(l,es);
		System.out.println(l);
		
		}

Map: ➢ Two column structure having key and value. ➢ Map is an interface. ➢ The classes which implement map are HasMap,TreeMap, HasTable. ➢ HasMap is for seraching ➢ TreeMap is for sorting **Initializaing a Map**		
➢ The key,value pair is specified along with the map. ➢ Similar to HasMap,TreeMap,HashSet and other classes also can be created.	Map<String,Integer> m=new HashMap<String, Integer>();	
Adding an Element ➢ The key and value are specified in the put method of map.	m.put("A",1); m.put("b",2);	

Traversing a Map This can be done in two ways:- ➢ Using KeySet ➢ Using EntrySet		

KeySet
 	
➢ Keys are fetchedand stored in an iterator.
➢ Then using the keys the values are got using the get method.
	private static void runusingkeyset(Map<String,Integer> m)
	{
	Set s=m.keySet();
	Iterator<String> itr=s.iterator();
	while(itr.hasNext())
	{
	String key=itr.next();
	Integer val=m.get(key);
	System.out.println("value is "+val);
	}
	}	

EntrySet
 	
➢ The key and value pair is together fetched using an iterator.
	private static void runusingenteryset(Map<String,Integer> m)
	{
	Set es=m.entrySet();
	Iterator<Map.Entry<String, Integer>> x= es.iterator();
	while(x.hasNext())
	{
	Map.Entry<String, Integer> y = x.next();
	System.out.println(y.getKey()
	+" -" +
	y.getValue());
	
	}
	
	}	

Get Method
 	
➢ Used for two things:- checking whether a key is present or not.
➢ Extracting a value specifying a key.
	private static void getele(String el, Map<String, Integer> m)
	{
	System.out.println("eleent is"+m.get(el));
	
	}
		Map.get(object);

Removing an Element	String key ="pencil";	

➢ The key is specified.
➢ Using the remove method of map the entry and value is removed.
	Integer temp = cart.remove(key);
	
	if(temp != null)
	System.out.println("entry removed");
	else
	System.out.println("key is not found");	

HashCode quickest way to find if two objects are different		
MULTITHREADING		
Starting a Thread Is used to start a new thread.It performs the following task:- ➢ A new thread is started. ➢ The thread moves from new state to runnable state. ➢ Its target run method is executed when it chance comes.		
This can be done in two ways :- ➢ By extending thread class. ➢ By implementing runnable interface.	t.start();	
1.)By Extending Thread class ➢ Class has to extend Thread class. ➢ Run method of Thread has to be overridden. ➢ Thread has to be started.	public class simplethread extends Thread { @Override public void run() { System.out.println("thread is running");	

	}
	public static void main(String[] args)
	{
	simplethread t=new simplethread();
	
	t.start();
	
	}
	}	

2.)By implementing runnable interface.
➢ We have a class which implements runnable interface.
➢ We override the run method.
➢ We create an object of the class;
➢ We create an object of thread class and pass the class object as parameter and then start the thread.
	public class Runnablethread implements Runnable
	{
	
	@Override
	public void run()
	{
	System.out.println("thread is running");
	//To change body of implemented methods use File | Settings | File Templates.
	}
	
	public static void main(String[] args)
	{
	Runnablethread r=new Runnablethread();
	Thread t=new Thread(r);
	t.start();
	}
	}	

Thread extending Vs Implementing runnable ➢ Implementing runnable preferred. ➢ Using `Runnable/Callable` gives you more flexibility that using Threads directly. ➢ Extends binds two class files very closely and can cause some pretty hard to		

deal with code. ➢ Henceforth threading will be demonstrated with runnable ➢ .Only extend Thread if you wish to modify the functionality of Threads.		
Knowing current running Thread.	Thread.currentThread()	

Performing Single Task with Multiple threads.
 	
Creating multiple instances of the same thread
	public class MultipleThreadsingleTAsk implements Runnable
	{
	@Override
	public void run()
	{
	System.out.println("thread is performing task");
	
	System.out.println(Thread.currentThread());
	}
	
	public static void main(String[] args)
	{
	MultipleThreadsingleTAsk m=new MultipleThreadsingleTAsk();
	Thread t=new Thread(m);
	Thread t1=new Thread(m);
	t.start();
	t1.start();
	}
	}	

Sleep:- ➢ Sleep is used to sleep a thread for a specific time. ➢ The thread scheduler picks another method when a thread is sleeping.	Thread.sleep();	

Stopping a Thread:
➢ Stopping a thread shouldnt be done suddenly
➢ Thread.stop() shouldn't be used anymore.
➢ Define our own method to stop a thread.
➢ Run method has to be changed accordingly
	public class Runnablethread
	implements Runnable
	{
	private boolean stopped=false;
	@Override
	public void run()
	{
	while(!stopped)
	{
	System.out.println("thread is running");
	}
	}
	public static void main(String[] args)
	{
	Runnablethread r=new Runnablethread();
	Thread t=new Thread(r);
	t.start();
	dosomejob();
	r.stop1();
	}
	private void stop1()
	{
	System.out.println("thread is about to be stopped");
	stopped=true;
	}
	private static void dosomejob()
	{
	try {
	Thread.sleep(1000);
	} catch (InterruptedException e)
	{
	e.printStackTrace();
	}
	}
	}	

Naming a thread:
➢ The method setname() is used name a thread.
➢ Getname() is usded to get the name of a thread.
	simplethread t=new simplethread();
	S.o.p("name of thread"+t.getName());
	t.setName("mythread");
	System.out.println("name of thread is "+t.getName());
		Threadobject.getname((;
		Threadobject.setname(“n”);

Synchronization in threads: ➢ Synchronization is the	public class Synchronization { synchronized void dosomejob(int n)	

capability to control the access of multiple threads to any shared resource.
➢ Better if we want one thread to access a resource at a time.
➢ Done to prevent thread interference.
➢ Consistency problems.
➢ Can be done to methods as well as blocks.
	{ int i;
	for (i=0;i<n;i++)
	{
	System.out.println("ans is"+n*i+" "+Thread.currentThread());
	try {
	Thread.sleep(400);
	} catch (InterruptedException e) {
	e.printStackTrace();
	}
	}
	}
	}	

Synchronized Blocks:
Only a particular block is synchronized.
	synchronized (this)
	{
	try
	{
	wait();
	} catch (InterruptedException e)
	{
	e.printStackTrace();
	}
	System.out.println("thread is about to resume");
	}	

Wait,Notify And Notifyall		
➢ **wait( )** tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls **notify( )**. ➢ **notify( )** wakes up the first thread that called **wait( )** on the same object. ➢ **notifyAll( )** wakes up all the threads that called **wait( )** on the same object. The highest priority thread		

will run first. ➢ Can be used only in synchronized methods		
Suspending and resuming a thread		

➢ Wait and notify is made use of here.
➢ System provided methods are depricated,
➢ Hence we use our own methods for suspending and resuming a tread.
	private boolean suspended=false;
	public void suspend1()
	{
	suspended=true;
	}
	public synchronized void resume1()
	{
	suspended=false;
	notify();
	}
	
	in run if thread is suspended:
	if(suspended)
	{
	System.out.println("thread is about to be suspended");
	synchronized (this)
	{
	ry
	{
	wait();
	} catch (InterruptedException e)
	{
	e.printStackTrace();
	}
	System.out.println("thread is about to resume");
	}
	}	

Timer and TimerTask		
Useful if we need to perform a task at intervals or repeatedly.		

➢ Implement a custom subclass of `TimerTask`. The `run` method contains	`public class Reminder {`
	`Timer timer;`
	
	`public Reminder(int seconds) {`	

the code that performs the task. In this example, the subclass is named RemindTask.
➢ Create a thread by instantiating the Timer class.
➢ Instantiate the TimerTask object (new RemindTask()).
➢ Schedule the timer task for execution.
	timer = new Timer();
	timer.schedule(new RemindTask(), seconds*1000);
	}
	
	class RemindTask extends TimerTask {
	public void run() {
	System.out.format("Time's up!%n");
	timer.cancel(); //Terminate the timer thread
	}
	}
	
	public static void main(String args[]) {
	new Reminder(5);
	System.out.format("Task scheduled.%n");
	}
	}	

Input and Output		
I/o is used to process the input and give an output based on the input. Java uses concept of streams to make I/o faster.		
<u>Stream</u> Stream is a sequence of data.In Java it is composed of bytes. Three availible streams in java:- System.in-standard input stream System.out-Standard output stream System.error-Standard error.		
➢ **Output Stream**-Used by java application to write data to a destination. ➢ Maybe a file,array,peripheral device or a socket ➢ **Input Stream**-Used by java application to write		

data to a destination. ➢ Maybe a file,array,peripheral device or a socket		

Serializable Interface:-
➢ In java serializable is mechanism thru which we make a copy of the object.
➢ The Serializableinterface is a marker interface your classes must implement if they are to be serialized / deserialized.
➢ Serializable is an empty interface.
➢ It indicates to the jvm to look at non static variable inside a class and make copy of it.
	public class emp implements Serializable
	{
	
	}	

Transient:- ➢ Sometimes you don't want to serialize/store all the fields in the object. ➢ To do this, you just need to declare a field as ***transient*** field.		
READING AND WRITING TEXT CONTENT		
Writing into a file:- ➢ The classes used are filewriter() and printwriter() ➢ Filewriter is the boss and printwriter is the assistant.(decorator design pattern)		

FileWriter() Class: Used to write character oriented data to the file.		
PrintWriter() class: ➢ The `PrintWriter` class enables you to write formatted data to an underlying `Writer`. ➢ For instance, writing `int`, `long` and other primtive data formatted as text, rather than as their byte values.		

Steps in writing data to a file:-
 	
➢ Create an instance of FileWriter and PrintWriter.
➢ Using printWriter object write data in to the required file.
➢ Close printwriter and filewriter instance in finally block.
➢ Check for exceptions.
	Main()
	{
	FileWriter fw=null;
	PrintWriter pw=null;
	fw=new FileWriter("a.exe",true);
	pw=new PrintWriter(fw);
	pw.println("IS it ok");
	pw.close();
	fw.close();
	System.out.println("program is fine");
	}
	
	◆ the above code doesn't handle exceptions	

Above code exception handling and finally block.	public static void main(String[] args)
	{
	FileWriter fw=null;
	PrintWriter pw=null;
	try {
	fw=new FileWriter("a.exe",true);
	
	pw=new PrintWriter(fw);
	pw.println("IS it ok");
	}
	catch (IOException e) {
	e.printStackTrace(); //To change body of catch statement use	

	File | Settings | File Templates.
	}
	
	finally
	{
	pw.close();
	if(fw!=null)
	{
	try {
	fw.close();
	} catch (IOException e) {
	e.printStackTrace();
	}
	}
	System.out.println("program is fine");
	}
	}	

Reading From A text File :- Filereader() and bufferedReader() classes are used.		
FileReader(): ➢ Used to read Data from a file.		
BufferedReader(): ➢ The BufferedReader class provides buffering to your Reader's. ➢ Buffering can speed up IO quite a bit. ➢ Rather than read one character at a time from the network or disk, you read a larger block at a time.		

Steps:
➢ Create an instance of FileReader and BufferedReader.
➢ Using BufferedReader object read data form required file.
	public static void main(String[] args)
	{
	FileReader fr=null;
	BufferedReader br=null;
	try {
	fr=new FileReader("a.exe");	

➢ Display the read data.
➢ Close BufferedReader. and FileReader instance in finally block.
➢ Check for exceptions.
	br=new BufferedReader(fr) ;
	String contents="";
	for(String msg="";
	msg=null;msg=br.readLine())
	{
	contents+=" \n " +msg;
	
	}
	System.out.println(contents);
	br.close();
	fr.close();
	}
	catch (FileNotFoundException e) {
	e.printStackTrace();
	} catch (IOException e) {
	e.printStackTrace();
	}
	}	

READING AND WRITING USER DEFINED OBJECTS		
Writing a user defined object into a file :- ➢ The classes used are FileOutputStream and ObjectOutputStream ➢ FileOutputStream is the boss and ObjectOutputStream is the assistant.(decorator design pattern)		
FileOutput Stream: It is an output stream to write data to a file. Can also be used with user defined objects.		
ObjectOutputStream:- ➢ The `ObjectOutputStream` class enables you to write Java objects to		

`OutputStream`'s instead of only bytes. ➢ You wrap an `OutputStream` in a `ObjectOutputStream` and then you can write objects to it.		

Steps:
 	
➢ Create an object of the user defined class which we want to pass.
➢ Create an instance of FileOutputStream and ObjectOutputStream.
➢ Using ObjectOutputStream object write user object into required file
➢ Close BufferedOutputReader. and FileOutputStream instance in finally block.
➢ Check for exceptions.
	public static void main(String[] args) {
	
	user u=new user();
	FileOutputStream fs= null;
	try {
	fs = new FileOutputStream("object.txt");
	ObjectOutputStream os=new ObjectOutputStream(fs);
	os.writeObject(u);
	os.close();
	fs.close();
	System.out.println("written successfully");
	}
	catch (FileNotFoundException e) {
	e.printStackTrace();
	}
	catch (IOException e) {
	e.printStackTrace();
	}
	}	

Reading a user defined Object from a file :- Writing a user defined object into a file :- ➢ The classes used are FileInputStream and ObjectInputStream ➢ FileInputStream is the boss and ObjectInputStream is the assistant.(decorator		

design pattern)		
FileInputStream class:- ➢ Obtains input bytes from a file. ➢ Used to read streams of data such as images.		
ObjectInputStream Class:- ➢ The `ObjectInputStream` class enables you to read Java objects from `InputStream`'s instead of only bytes. ➢ You wrap an `InputStream` in a `ObjectInputStream` and then you can read objects from it		

Steps:
➢ Create an object of the user defined class and assign it to null.
➢ Create an instance of FileOutInStream and ObjectInputStream.
➢ Using ObjectInputStream object read the data from file and assign it to user object.
➢ Explicit casting necessary.
➢ Close BufferedInputReader. and FileInputStream instance in finally block.
➢ Check for exceptions.
	public static void main(String[] args) {
	
	user o=null;
	try {
	FileInputStream fis=new FileInputStream("object.txt");
	ObjectInputStream ois=new ObjectInputStream(fis);
	o= (user) ois.readObject();
	ois.close();
	System.out.println("data read successfully"+o);
	
	fis.close();
	}
	catch (IOException e) {
	e.printStackTrace(); .
	}
	catch (ClassNotFoundException e) {
	e.printStackTrace();
	}
	}	

Strings		
➢ String is an immutable	`String t="rohan";`	

object in java. ➢ String can be created in 2 ways.: ➢ Using new keyword-object created in non pool memory region ➢ Directly-object created in shared pool region.One memory region for an object	String t=”rohan”; String a=new string(“lava”); String a=new string(“lava”); t.concat(" mudaliar"); System.out.println(t); Output:rohan ▪ The reference has to be given i.e T=t.concat(“mudaliar”); ▪ Only single object created for “rohan” ▪ Two objects created for “lava”.	

String comparision
Three methods :-
➢ Using equasl() method and equalsIgnorecase()
➢ Using = =
➢ Using compareto() – return an integer value,
If s1==s2:-0 returned
If s1>s2 :+ve value
If s1<s2:-ve value
	private static void comparisonstring()
	{
	String a="a";
	String b="a";
	String c=new String("a");
	String d="A";
	
	System.out.println(a==b);//true
	
	System.out.println(a==c);//false
	
	System.out.println(a==d);//false
	
	System.out.println(a.equals(d));//false
	
	System.out.println(a.equalsIgnoreCase(d));//true
	
	System.out.println(a.compareTo(d));//0 or 1 returned
	
	System.out.println(a.compareToIgnoreCase(c));
	
	}	

String concatenation Done using : +-the compiler converts it into string builder. Concat method-string appended at end	String a="ok"; String b="fine"; String c=a+b+1; System.out.println(c); ▪ a total of 3 objects are created in the above case	

	for a,b and c ▪ Usage of string builder in case of concatenation different data types in recomemded	

Substring:
Can be got using:
➢ Public String substrint(int startindex)
➢ Public String substrint(int endindex)
	`String t="rohan mudaliar"`
	`System.out.println(t.substring(1));//`
	
	`System.out.println(t.substring(1, 6));`
	
	Output:
	ohan mudaliar
	ohan	

String buffer: It is used to create a mutable version of string It is thread safe.		
Networking		
Concept of connecting two or more computers so that resources can be shared.		
Ip address-unique no assigned to a node of network Protocols-A set of rules to be followed for communication.		
Socket Programing: ➢ Socket programming is performed for communication between 2 computers. ➢ For connection oriented server and server socket class is used. ➢ Two information is needed for connection oriented socket programing.(1.)Ip address of server(2.)Portno		
Socket class: Acts as an end point between 2 machines.	`public class Myserver` {	

Used to create a socket.
Commonly used methods are:
InputSteram getInputstream();
OutputStrem getOutputstream();
Synchronized Void close()
All above methods are public.
Server class generates `IOException`
	`public static void main(String[] args) { ServerSocket ss;`
	
	`ss = new ServerSocket(5015); Socket s=ss.accept(); DataInputStream dis=new DataInputStream(s.getInputStream()); String xyz=dis.readUTF(); System.out.println("msg is"+xyz); ss.close();`
	
	`}`	

ServerSocket Used to create a server socket. Establishes communication btwn clients. Methods are Socket accept(); InputSteram getInputstream(); OutputStrem getOutputstream(); Synchronized Void close(); All above methods are public. Client class is given.It gives `IOException, UnknownHostException`	`Socket s=new Socket("localhost",5015); DataOutputStream dos=new DataOutputStream(s.getOutputStream()); dos.writeUTF("hello hi test"); dos.flush(); dos.close(); s.close();`	

Url class
Points to a resource on the world wide web.
Common methods are:
getPortno();
getHost();
getProtocol();
getfile()
All of them return string.
`MalformedURLException got.`
	`URL url=new URL("http://www.javatpoint.com"); System.out.println("protocol"+url.getProtocol()); System.out.println("host"+url.getHost()); System.out.println("portno"+url.getPort());`	

Inet class
Provides methods to get ip of any hostname
	`InetAddress ip=InetAddress.getByName("www.javatpoint.com");`
	
	`System.out.println("Host Name: "+ip.getHostName()); System.out.println("IP Address: "+ip.getHostAddress());`	

DatagramSocket class Represents a connectionless	import java.net.DatagramPacket; import java.net.DatagramSocket;	import java.net.DatagramPacket;

Socket for sending and receiving datagram packets.	public class DReceiver{
	  public static void main(String[] args) throws Exception {
	
	  DatagramSocket ds = new DatagramSocket(3000);
	  byte[] buf = new byte[1024];
	
	  DatagramPacket dp = new DatagramPacket(buf, 1024);
	  ds.receive(dp);
	
	  String strRecv = new String(dp.getData(), 0, dp.getLength());
	  System.out.println(strRecv);
	  ds.close();
	  }
	}
		import java.net.DatagramSocket;
		import java.net.InetAddress;
		
		public class DSender{
		  public static void main(String[] args) throws Exception {
		  DatagramSocket ds = new DatagramSocket();
		  String str = "hello world";
		  InetAddress ia = InetAddress.getByName("127.0.0.1");
		  DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ia, 3000);
		  ds.send(dp);
		  ds.close();
		  }
		}