Java tutorial's

Class & Objects

1) Static variable & method

```
public class TestStaticVar{
    static String name = "Aditya";
    static String city = "Banglore";

    static public void methodS(){
        System.out.println("Accessing var using static method : "+name + " "+city);
        city = "Patna";
        System.out.println("modifying var data : "+city);
    }
    public static void main(String[] args) {
        methodS();
    }

}
```

Output

```
Accessing var using static method : Aditya Banglore
modifying var data : Patna
```

2) String is immutable

```
class TestStringObject{
    public static void main(String[] args){
        String str1 = "First string";
        System.out.println(str1);
        System.out.println(str1.hashCode());

        str1 = "Modified string";
        System.out.println(str1);
        System.out.println(str1.hashCode());
        // from this we come to the conclusion that strings are immutable

    }
}
```

Output

```
First string
158852001
Modified string
1225095464
```

3) Passing Array as Method

```
public class PassArrayAsMethod{
    static void findMin(int[] ar){
        int sum=0;
        for(int i=0;i<ar.length;i++){
            sum=sum+ar[i];
        }
        System.out.println(sum);

    }
```

```
    public static void main(String[] args) {
        PassArrayAsMethod obj = new PassArrayAsMethod();
        int a[]={11,22,33,44,55};
        findMin(a);

    }
}
```

Output

```
    Array sum : 165
```

4）**Class object**

```
class Test{
    public void displayC1(){
        System.out.println("Class First method");
    }
}

class TestClass2{
    public void dsiplayC2(){
        Test obj3 = new Test();
        obj3.displayC1();
        System.out.println("Class second method");
    }
}

class TestCLassObj{
    public static void main(String[] args) {
        TestClass2 obj2 = new TestClass2();
          obj2.dsiplayC2();
    }
}
```

Output

```
    Class First method
    Class second method
```

5）Blocks in Java

```
package BasicAbstract;

abstract class CheckBlocks {
    CheckBlocks(){
        System.out.println("Abstract class constructor");
    }
    {
        System.out.println("Instance block ");
    }
    static    {
        System.out.println("Static block");
    }
}
class Test4 extends CheckBlocks{
    Test4(){
        super();
        System.out.println("Normal class constructor");
```

```
    }
    public static void main(String[] args)
    {
        new Test4();
        new Test4();
    }
}
```

Output

```
    Static block
    Instance block
    Abstract class constructor
    Normal class constructor
    Instance block
    Abstract class constructor
    Normal class constructor
```

6）**Abstract class**

```
abstract class Abs {
    abstract void m1();
    abstract void m2();
    abstract void m3();
    void m4() {
        System.out.println("m4 method");
    }
}
class Test extends Abs {
    void m1() {
        System.out.println("m1 method");
    }
    void m2() {
        System.out.println("m2 method");
    }
    void m3() {
        System.out.println("m3 method");
    }
    public static void main(String[] args)
    {
        Test obj=new Test();
        obj.m1();    obj.m2();    obj.m3();    obj.m4();
        Abs obj1=new Test();
        obj1.m1();    // compile time-- Abs(parent) checked && runtime -- test(child) executed
        obj1.m2();    // compile time-- Abs(parent) checked && runtime -- test(child) executed
        obj1.m3();    // compile time-- Abs(parent) checked && runtime -- test(child) executed
        obj1.m4();        // compile time-- Abs(parent) checked && runtime -- test(child) executed

    }
}
```

Output

```
    m1 method m2 method m3 method m4 method
    m1 method m2 method m3 method m4 method
```

Inheritance :-

---

1) Single Inheritance

```
class Vehicle{
	void print(){
		System.out.println("base class");
	}
}
class Single extends Vehicle{
	void print(){
		super.print();	// super method to call parent method
		System.out.println("from child");
	}
	void dif(){
		System.out.println("new child method");
	}
	public static void main(String[] args){
	Single s=new Single();
	s.print();
	s.dif();
}
}
```

Output

```
	base class
	from child
	new child method
```

---

2) Super class constructor

```
class Superclass{
	int age;
	Superclass(int age){
	this.age = age;
	}
	public void getAge(){
	System.out.println("The value of the variable named age in super class is: " +age);
	}
}
class SuperClassConstructor extends Superclass {
	SuperClassConstructor(int age){
	super(age);
	}
	public static void main(String argd[]){
		SuperClassConstructor s = new SuperClassConstructor(24);
		s.getAge();
		}
	}
```

Output

```
		The value of the variable named age in super class is: 24
```

---

3) Hierarchical Inheritance

```
class Animal{
	public void m1(){
		System.out.println("Animal parent class");
	}
}
class Mammal extends Animal{
	public void m2(){
		super.m1();
		System.out.println("Method from mammal class");
	}
}
class Reptile extends Animal{
	public void m3(){
	System.out.println("Method from reptile class ");
	}
}
public class MultiLevel{
public static void main(String args[]){
	Mammal m = new Mammal();
	Reptile d = new Reptile();
	m.m2(); d.m3();
	}
}
```

Output

```
Animal parent class
Method from mammal class
Method from reptile class
```

4) Override

```
class Parent {
	public void method1(){
		System.out.println("Parent method");
	}
}
class OverRide extends Parent{
	public void method1(){
		super.method1();
		System.out.println("Child method");
	}
	public static void main(String[] args) {
		Parent obj = new OverRide();
		obj.method1();
	}
}
```

Output

```
Parent method
Child method
```

5) Interface (multiple interface)

```
interface It1 {    //abstract 100%
      void m1();    //public abstract
      void m2();    //public abstract
      void m3();    //public abstract
 }
class Test implements It1 {      //implementation of interface It1
      public void m1() {
            System.out.println("M1 method");
      }
      public void m2() {
            System.out.println("M2 method");
      }
      public void m3() {
            System.out.println("M3 method");
      }
      public static void main(String[] args)
      {
            Test obj=new Test();
            obj.m1();       obj.m2();      obj.m3();

            It1 i=new Test();    //interface id variable is able to hold the implementation class object.
            i.m1(); i.m2(); i.m3();
      }
}
```

Output

```
            M1 method
            M2 method
            M3 method
```

---

6) Interface extends relation

```
package itextends;

interface It1 {
      void m1();
}
interface It2 extends It1 {
      void m2();
}
interface It3 extends It2 {
      void m3();
}

class Test implements It1,It2,It3 {
      @Override
      public void m1() {
            System.out.println("M1 method--1");
      }

      @Override
      public void m2() {
```

```
        System.out.println("M2 method--2");
    }
    public void m3() {
        System.out.println("M3 method--3");
    }
    public static void main(String[] args)
    {
        Test obj=new Test();
        obj.m1();       obj.m2();       obj.m3();
    }
}
```

Output

```
        M1 method--1
        M2 method--2
        M3 method--3
```

---

7) Garbage Collector (gc)

```
// Garbage collector...   System class call gc (static method) System.gc
class GarbageDemo {
    int x=10;
    int y=20;

    public void finalize() throws Throwable {
      System.out.println("object delete ");
    }
    static void show() {
        GarbageDemo obj = new GarbageDemo();
    }
    static void get() {
        GarbageDemo obj1 = new GarbageDemo();
        obj1.show();
    }
    public static void main(String[] args)
    {
        get();
        GarbageDemo.show();
        for(int i =1; i<=10; i++){
        System.gc();
        try{
            Thread.sleep(1000);
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
    }
}
```

Output

```
    object delete
    object delete
    object delete
```

---

8) Exception Handling

```
import java.util.*;
class TestThrow{

	static void validate(int age){
		if(age<18){
			throw new ArithmeticException("Not valid");
		}else{
			System.out.println("welcome to vote");
		}
	}
	public static void main(String[] args) {
		try{
			Scanner sc = new Scanner(System.in);
			System.out.println("Enter age");
			int a= sc.nextInt();
			validate(a);
		}catch(ArithmeticException e){
			System.out.println("error : "+e);
		}
		System.out.println("rest of code");
	}
}
```

Output

```
	Enter age
	20
	welcome to vote
	rest of code
```

---

9) Unchecked exception & nesting of try

```
class Test2 {
public static void main(String[] args)
{
	try {                                          // nesting of try block
                                                       // every catch block can only execute after the try block
		try {
			int x=10/0;
		}
		catch (ArithmeticException e)
	{
		System.out.println(e);
	}
		int[] y =new int[5];
		y[10]=20;
	}
	catch(ArrayIndexOutOfBoundsException a)
	{
		System.out.println(a);
	}
   }
}
```

Output

```
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
```

---

10) Multiple catch

```
class MultipleCatch{
    public static void main(String[] args) {
        int[] num={4,8,16,32,64,128};
        int[] denum={2,0,4,4,0};
        for(int i=0;i<num.length;i++)
            try{
            System.out.println(num[i]+"/"+denum[i]+"="+ (num[i]/denum[i]));
        }catch(ArithmeticException e){
            System.out.println("error 1 catch : "+e);
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println("error 2 catch");
        }
        System.out.println("rest of code");
    }

}
```

Output

```
4/2=2
error 1 catch : java.lang.ArithmeticException: / by zero
16/4=4
32/4=8
error 1 catch : java.lang.ArithmeticException: / by zero
error 2 catch
rest of code
```

---

11)