

```
class Solution {  
    public String HelloWorld() {  
        return "Hello World";  
    }  
}  
  
import java.util.Scanner;
```

Q. What are we doing here?
ans: For a method of class Solution we're making an object to call that method by putting input parameters in it (here, void). Also we're printing that object.method() since in method we're returning the answer

```
public class Main {  
    public static void main(String[] args) {  
        Scanner scanner = new  
Scanner(System.in);  
        Solution solution = new Solution();  
        System.out.println(solution.HelloWo  
rld());  
    }  
}
```

```
15 String name;  
16 int age;  
17  
18 public void printInfo() {  
19     System.out.println(this.name);  
20     System.out.println(this.age);  
21 }  
22  
23 Student(Student s2) {  
24     this.name = s2.name;  
25     this.age = s2.age;  
26 }  
27  
28 Student() {  
29  
30 }  
31 }  
32  
33 public class OOPS {  
34     Run | Debug  
35     public static void main(String args[]) {  
36         Student s1 = new Student();  
37         s1.name = "aman";  
38         s1.age = 24;
```




```
17     public void printInfo(String name) {
18         System.out.println(name);
19     }
20
21
22     public void printInfo(int age) {
23         System.out.println(age);
24     }
25
26     public void printInfo(String name, int age) {
27         System.out.println(name + " " + age);
28     }
29 }
30
31 public class OOPS {
32     Run | Debug
33     public static void main(String args[]) {
34         Student s1 = new Student();
35         s1.name = "aman";
36         s1.age = 24;
37
38         s1.printInfo(s1.name, s1.age);
39     }
}
```



```
1  class Shape {
2      public void area() {
3          System.out.println("displays area");
4      }
5  }
6
7  class Triangle extends Shape {
8      public void area(int l, int h) {
9          System.out.println(1/2*l*h);
10     }
11 }
12
13 class EquilateralTriangle extends Triangle {
14     public void area(int l, int h) {
15         System.out.println(1/2*l*h);
16     }
17 }
18
19 public class OOPS {
20     Run | Debug
21     public static void main(String args[]) {
22     }
23 }
```

```
1  class Shape {
2      public void area() {
3          System.out.println("displays area");
4      }
5  }
6
7  class Triangle extends Shape {
8      public void area(int l, int h) {
9          System.out.println(1/2*l*h);
10     }
11 }
12
13 class Circle extends Shape {
14     public void area(int r) {
15         System.out.println((3.14)*r*r);
16     }
17 }
18
19
20 public class OOPS {
21     Run | Debug
22     public static void main(String args[]) {
23
24     }
```





```
1 package bank;
2
3 class Account {
4     public String name;
5 }
6
7 public class Bank {
8
9 }
```




```
16 class Circle extends Shape {
17     public void area(int r) {
18         System.out.println((3.14)*r*r);
19     }
20 }
21
22
23 public class OOPS {
    Run | Debug
24     public static void main(String args[]) {
25         bank.Account account1 = new bank.Account();
26         account1.name = "customer1";
27     }
28 }
```






```
1 import java.util.*;
2 import bank;
3
4 class Shape {
5     public void area() {
6         System.out.println("displays area");
7     }
8 }
9
10 class Triangle extends Shape {
11     public void area(int l, int h) {
12         System.out.println(1/2*l*h);
13     }
14 }
15
16 class Circle extends Shape {
17     public void area(int r) {
18         System.out.println((3.14)*r*r);
19     }
20 }
21
22
23 public class OOPS {
```

Run | Debug





```
1 package bank;
2
3 class Account {
4     public String name;
5     protected String email;
6     private String password;
7 }
8
9 public class Bank {
10     Run | Debug
11     public static void main(String args[]) {
12         Account account1 = new Account();
13         account1.name = "Apna College";
14         account1.email = "apnacollege@gmail.com";
15         account1.password = "abcd";
16     }
```



```
5      protected String email;
6      private String password;
7
8      //getters & setters
9      public String getPassword() {
10         return this.password;
11     }
12
13     public void setPassword(String pass) {
14         this.password = pass;
15     }
16 }
17
18 public class Bank {
19     Run | Debug
20     public static void main(String args[]) {
21         Account account1 = new Account();
22         account1.name = "Apna College";
23         account1.email = "apnacollege@gmail.com";
24         account1.password = "abcd";
25     }
26 }
```



```
8 //getters & setters
9 public String getPassword() {
10     return this.password;
11 }
12
13 public void setPassword(String pass) {
14     this.password = pass;
15 }
16 }
17
18 public class Bank {
19     Run | Debug
20     public static void main(String args[]) {
21         Account account1 = new Account();
22         account1.name = "Apna College";
23         account1.email = "apnacollege@gmail.com";
24         account1.setPassword("abcd");
25         System.out.println(account1.getPassword());
26     }
27 }
```



```
1  abstract class Animal {
2      |   abstract void walk();
3      |   }
4
5      class Horse extends Animal {
6      |   public void walk() {
7      |       |   System.out.println("Walks on 4 legs");
8      |       |   }
9      |   }
10
11     class Chicken extends Animal {
12     |   public void walk() {
13     |       |   System.out.println("Walks on 2 legs");
14     |       |   }
15     |   }
16
17     public class OOPS {
18     |       Run | Debug
19     |       public static void main(String args[]) {
20     |           |   Horse horse = new Horse();
21     |           |   horse.walk();
22     |       |   }
23     |   }
```



```
1  abstract class Animal {
2      ⚡ abstract void walk();
3      ... public void eat() {
4          ... System.out.println("Animal eats");
5      }
6  }
7
8  class Horse extends Animal {
9      public void walk() {
10         System.out.println("Walks on 4 legs");
11     }
12 }
13
14 class Chicken extends Animal {
15     public void walk() {
16         System.out.println("Walks on 2 legs");
17     }
18 }
19
20 public class OOPS {
21     Run | Debug
22     public static void main(String args[]) {
23         Horse horse = new Horse();
24         horse.walk();
25     }
26 }
```



```
8 class Horse extends Animal {
9     public void walk() {
10         System.out.println("Walks on 4 legs");
11     }
12 }
13
14 class Chicken extends Animal {
15     public void walk() {
16         System.out.println("Walks on 2 legs");
17     }
18 }
19
20 public class OOPS {
21     Run | Debug
22     public static void main(String args[]) {
23         Horse horse = new Horse();
24         horse.walk();
25         horse.eat();
26     }
27 }
```



- Abstraction is achieved in 2 ways :
- Abstract class
 - Interfaces (Pure Abstraction)

1. Abstract Class

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

```
abstract class Animal {  
    abstract void walk();  
}
```




```
10
11 class Horse extends Animal {
12     Horse() {
13         System.out.println("Created a Horse");
14     }
15     public void walk() {
16         System.out.println("Walks on 4 legs");
17     }
18 }
19
20 class Chicken extends Animal {
21     public void walk() {
22         System.out.println("Walks on 2 legs");
23     }
24 }
25
26 public class OOPS {
27     Run | Debug
28     public static void main(String args[]) {
29         Horse horse = new Horse();
30     }
31 }
```



```
1 interface Animal {  
2     public void walk();  
3 }  
4  
5 class Horse implements Animal {  
6     public void walk() {  
7         System.out.println("walks on 4 legs");  
8     }  
9 }  
10  
11 public class OOPS {  
12     Run | Debug  
13     public static void main(String args[]) {  
14         Horse horse = new Horse();  
15         horse.walk();  
16     }  
17 }
```



2. Interfaces


- All the fields in interfaces are public, static and final by default.
- All methods are public & abstract by default.
- A class that implements an interface must implement all the methods declared in the interface.
- Interfaces support the functionality of multiple inheritance.

```
interface Animal {  
  
    void walk();  
  
}  
  
class Horse implements Animal {
```

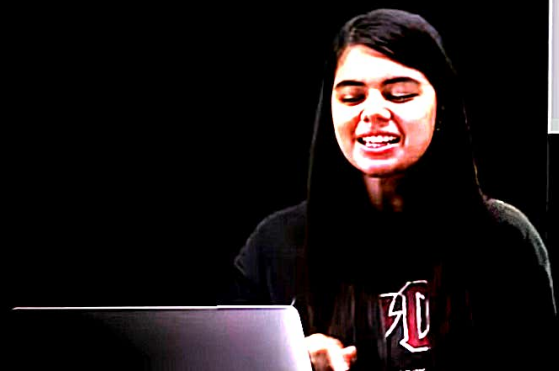



```
1  interface Animal {
2      int eyes = 2;
3      void walk();
4  }
5
6  interface Herbivore {
7
8  }
9
10 class Horse implements Animal, Herbivore {
11     public void walk() {
12         System.out.println("walks on 4 legs");
13     }
14 }
15
16 public class OOPS {
17     Run | Debug
18     public static void main(String args[]) {
19         Horse horse = new Horse();
20         horse.walk();
21     }
22 }
```






```
1 class Student {  
2     String name;  
3     static String school;  
4 }  
5  
6 public class OOPS {  
    Run | Debug  
7     public static void main(String args[]) {  
8         Student.school = "JMV";  
9     }  
10 }
```





```
1 class Student {  
2     String name;  
3     static String school;  
4 }  
5  
6 public class OOPS {  
    Run | Debug  
7     public static void main(String args[]) {  
8         Student.school = "ABC";  
9         Student student1 = new Student();  
10        student1.name = "tony";  
11        System.out.println(student1.school);  
12    }  
13 }
```





```
1  class Student {  
2      String name;  
3      static String school;  
4  
5      public static void changeSchool() {  
6          school = "newschool";  
7      }  
8  }  
9  
10 public class OOPS {  
11     Run | Debug  
12     public static void main(String args[]) {  
13         Student.school = "ABC";  
14         Student student1 = new Student();  
15         student1.name = "tony";  
16         System.out.println(student1.school);  
17     }  
18 }
```

