Abstraction

Abdul Haseeb Shaikh

Abstraction

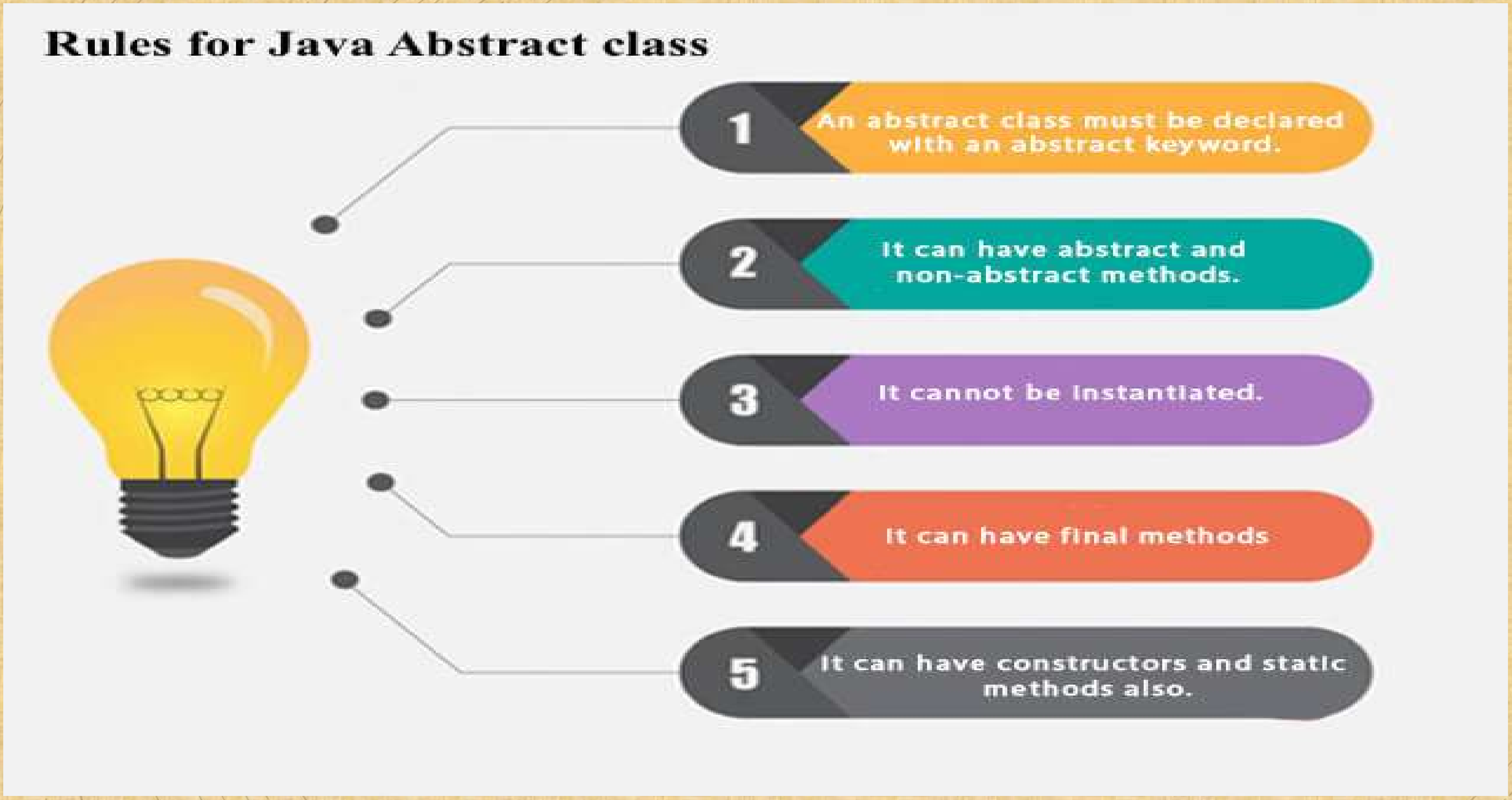
- **Abstraction** is a process of hiding the implementation details (information hiding) and showing only functionality to the user.
- Focus is on:
  - What an object does rather than how it does that!

How to achieve Abstraction?

- Abstract Classes
- Interfaces

Abstract Classes

- Class Declared with abstract keyword



Rules for Java Abstract class

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.
4. It can have final methods
5. It can have constructors and static methods also.

Reference: javatpoint

Abstract Classes

- A generalized class (Abstract):
  - Shared by all subclasses
  - Subclasses fill in the detail according to their own wish. (Overriding)
  - Declared with the help of abstract keyword
  - Can’t be instantiated:
    - Useless as it isn’t fully defined
  - Can’t declare abstract constructors:
    - Can’t override constructor’s, only overload them
  - Can’t declare abstract static methods:
    - Static prevents overriding

Abstract Classes

- If a class is extending an Abstract class:
  - It must provide implementation of all of its abstract methods
- Otherwise declare that class as an Abstract class

Abstract Method

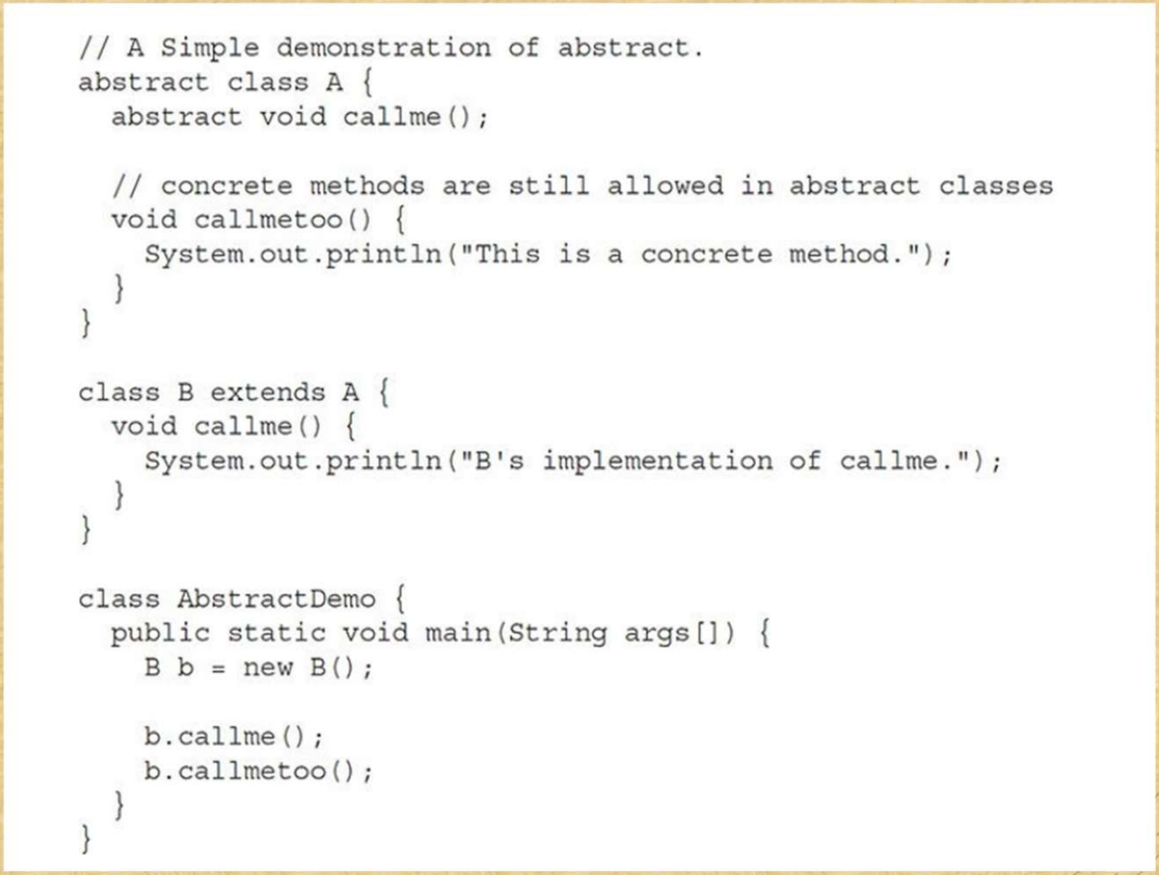
- In super classes you can not provide that much of meaning to methods
- Consider figure class having area() method?
- But in triangle it will have perfect meaning
- So we need methods that must be overridden by subclasses in order to have meaning:
  - Abstract Methods

Abstract Method

- Any method with abstract keyword
- It has no body in parent class, where it is defined
- Subclass must override its implementation
  - Subclasser responsibility
- Abstract methods can reside only in abstract class

```
abstract type name(parameter-list);
```

Demo

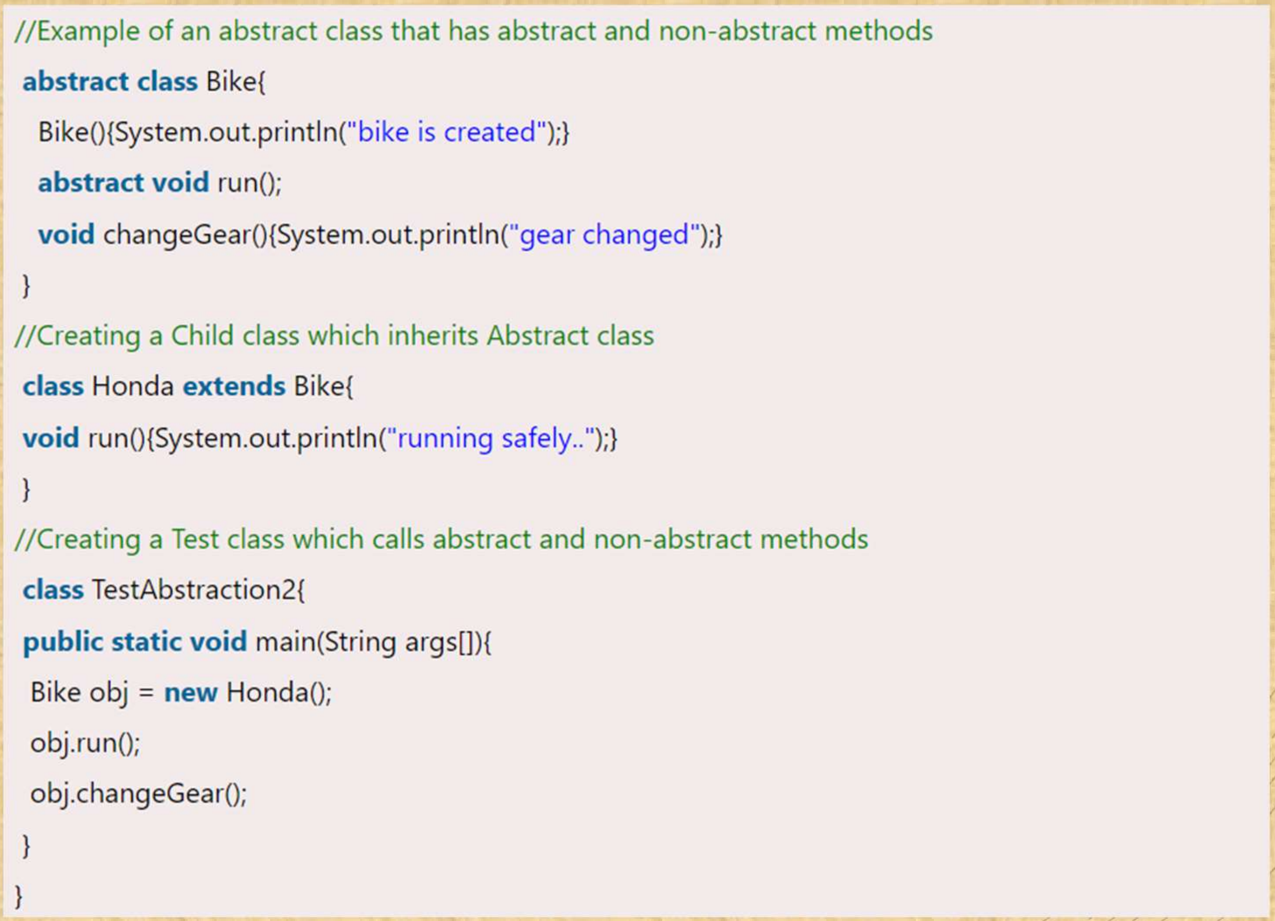


```
// A Simple demonstration of abstract.
abstract class A {
  abstract void callme();

  // concrete methods are still allowed in abstract classes
  void callmetoo() {
    System.out.println("This is a concrete method.");
  }
}

class B extends A {
  void callme() {
    System.out.println("B's implementation of callme.");
  }
}

class AbstractDemo {
  public static void main(String args[]) {
    B b = new B();

    b.callme();
    b.callmetoo();
  }
}
```

Concrete Class vs Abstract Class

- Concrete class is fully defined and has concrete methods (methods with definition)
- Abstract class has abstract methods, and they have partial or no Implementation

Example, containing abstract/non abstract and constructor

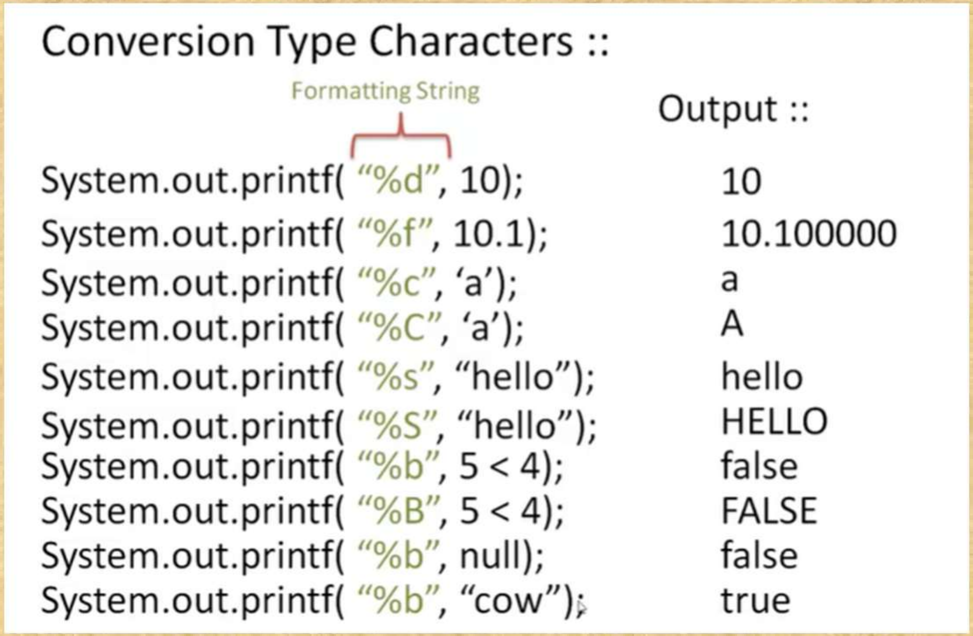


```
//Example of an abstract class that has abstract and non-abstract methods
 abstract class Bike{
   Bike(){System.out.println("bike is created");}
   abstract void run();
   void changeGear(){System.out.println("gear changed");}
 }
//Creating a Child class which inherits Abstract class
 class Honda extends Bike{
 void run(){System.out.println("running safely..");}
 }
//Creating a Test class which calls abstract and non-abstract methods
 class TestAbstraction2{
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
```

Why Constructor?

- Because:
  - Abstract classes can have variables of all types
  - They need to be initialized to default values
    - Constructors become necessary then

System.out.printf()



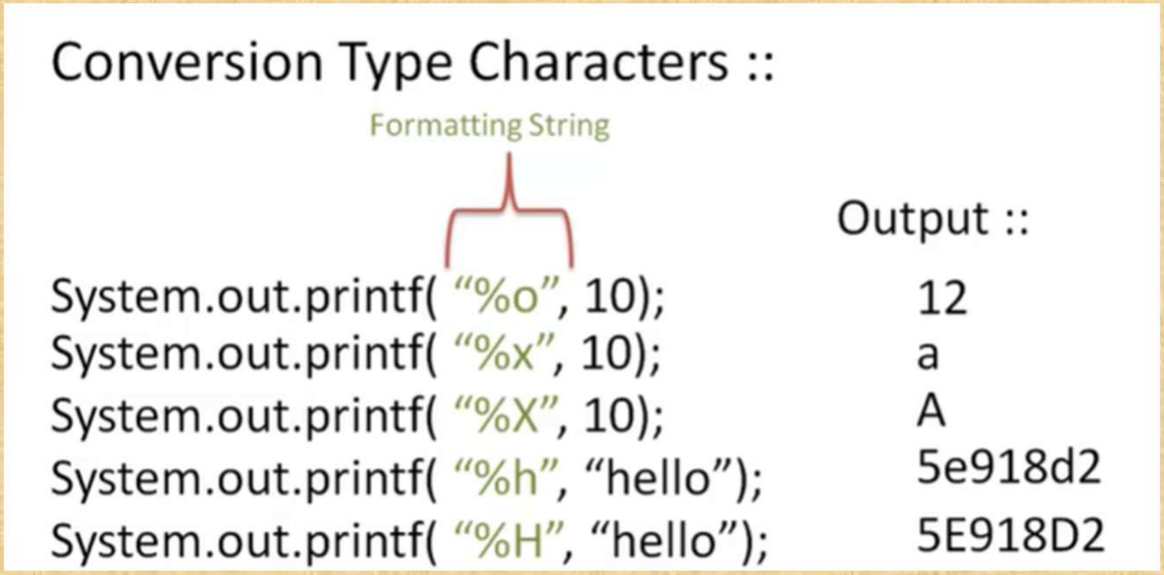
Conversion Type Characters ::

Formatting String

```
System.out.printf( “%d”, 10);              10
System.out.printf( “%f”, 10.1);            10.100000
System.out.printf( “%c”, ‘a’);             a
System.out.printf( “%C”, ‘a’);             A
System.out.printf( “%s”, “hello”);         hello
System.out.printf( “%S”, “hello”);         HELLO
System.out.printf( “%b”, 5 < 4);           false
System.out.printf( “%B”, 5 < 4);           FALSE
System.out.printf( “%b”, null);            false
System.out.printf( “%b”, “cow”);           true
```

Output ::

System.out.printf()

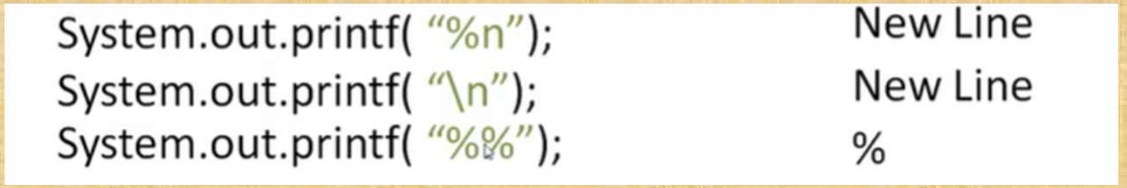


Conversion Type Characters ::

Formatting String

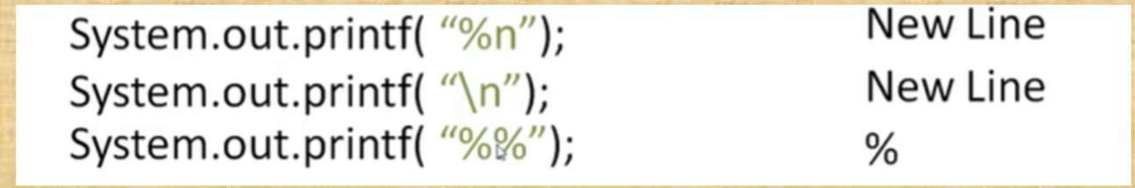
```
                                              Output ::
System.out.printf( "%o", 10);                    12
System.out.printf( "%x", 10);                    a
System.out.printf( "%X", 10);                    A
System.out.printf( "%h", "hello");               5e918d2
System.out.printf( "%H", "hello");               5E918D2
```

System.out.printf()

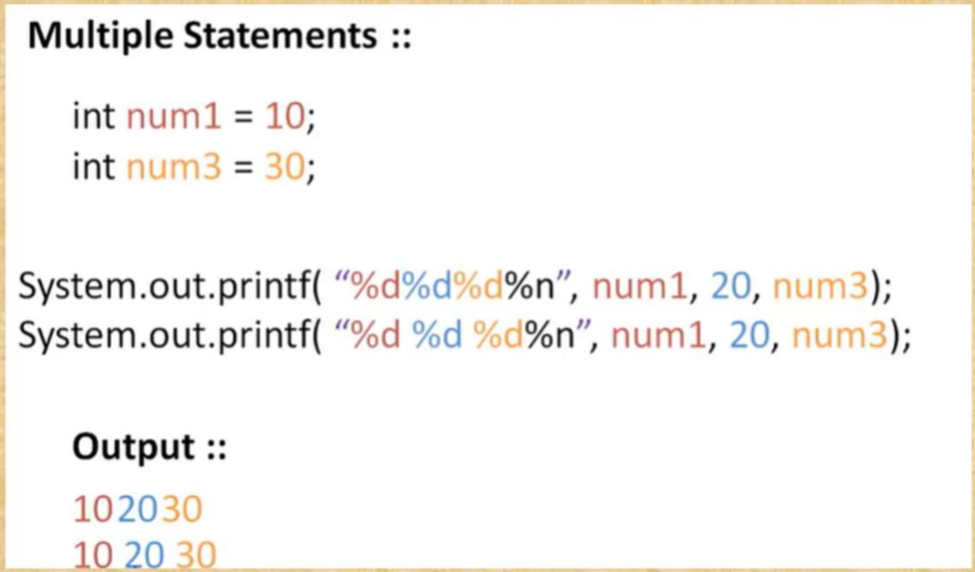


`System.out.printf( "%n");`	New Line
`System.out.printf( "\n");`	New Line
`System.out.printf( "%%");`	%

System.out.printf()



```
System.out.printf( "%n");          New Line
System.out.printf( "\n");          New Line
System.out.printf( "%%");          %
```



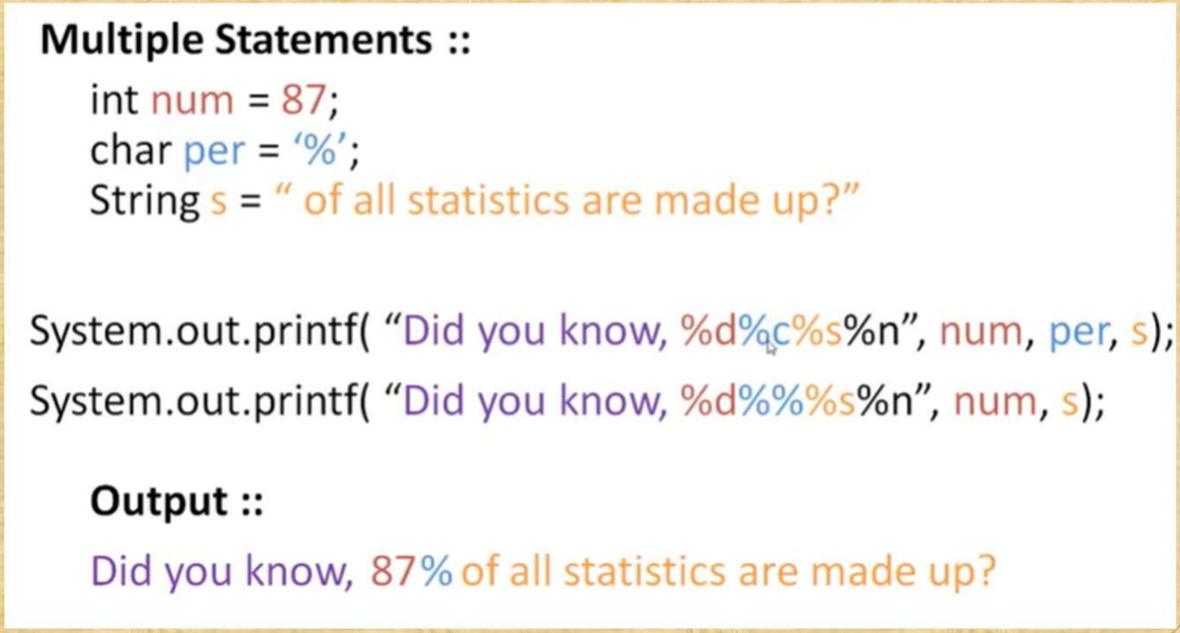
Multiple Statements ::

```
    int num1 = 10;
    int num3 = 30;


System.out.printf( "%d%d%d%n", num1, 20, num3);
System.out.printf( "%d %d %d%n", num1, 20, num3);
```

Output ::

```
102030
10 20 30
```

System.out.printf()



Multiple Statements ::

```
    int num = 87;
    char per = '%';
    String s = " of all statistics are made up?"


System.out.printf( "Did you know, %d%c%s%n", num, per, s);
System.out.printf( "Did you know, %d%%%s%n", num, s);
```

Output ::

```
Did you know, 87% of all statistics are made up?
```