edureka!

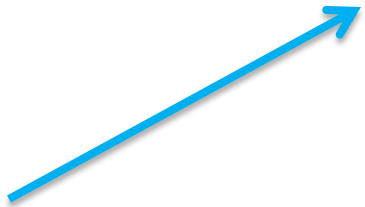
Java Constructors

A **constructor :**

- is used in the creation of an object.
- is a special method with no return type.
- must have the *same name as the class it is in.*
- is used to initialize the object.
- if not defined will initialize all instance variables to default value.

Constructor

```
public class Class2 {

 int x;
 int y;

 public Class2() {
  x = 10;
  y = 11;
 }
}
```



```
public class Class1 {

 public static void main(String[] args) {
  Class2 ob = new Class2();

  System.out.println("Value of x when
     constructor is called: " + ob.x);
  System.out.println("Value of y when
     constructor is called: " + ob.y);
  }
}
```

eureka!

This keyword

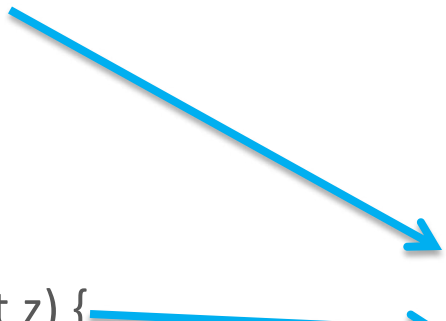
edureka!

- **this** is a keyword used to reference the current object within an instance method or a constructor.

Constructor Overloading

```
public class Class2 {
 int x, y;
 public Class2() {
  x = 10;
  y = 11;
 }
 public Class2(int z) {
  x = z;
  y = z;
 }
}
```

```
public class Class1 {

 public static void main(String[]
    args) {

   Class2 ob = new Class2();
   Class2 ob = new Class2(5);

 }
```



eureka!

edureka!

Java Review (Packages)

What is Package?

edureka!

- A **Java package** is a mechanism for organizing **Java classes** into **namespaces**
- Programmers use packages to **organize classes** belonging to the same category
- Classes in the same package can access each other's package-access members

Advantage of a Package

edureka!

- Programmers can easily determine that these classes are related
- Programmers know where to find files of similar types
- The names won't conflict
- You can have define access of the types within the package

Naming Convention of a Package

edureka!

- Package names are written in all **lower case**

- Companies use their **reversed Internet domain** name to begin their package names—

  for example, **com.example.mypackage** for a package named **mypackage** created by a programmer at **example.com**

Naming Convention of a Package

edureka!

If the domain name contains -

i. a hyphen or a **special character**

ii. if the package name begins with **a digit, illegal character reserved Java keyword such as "int“**

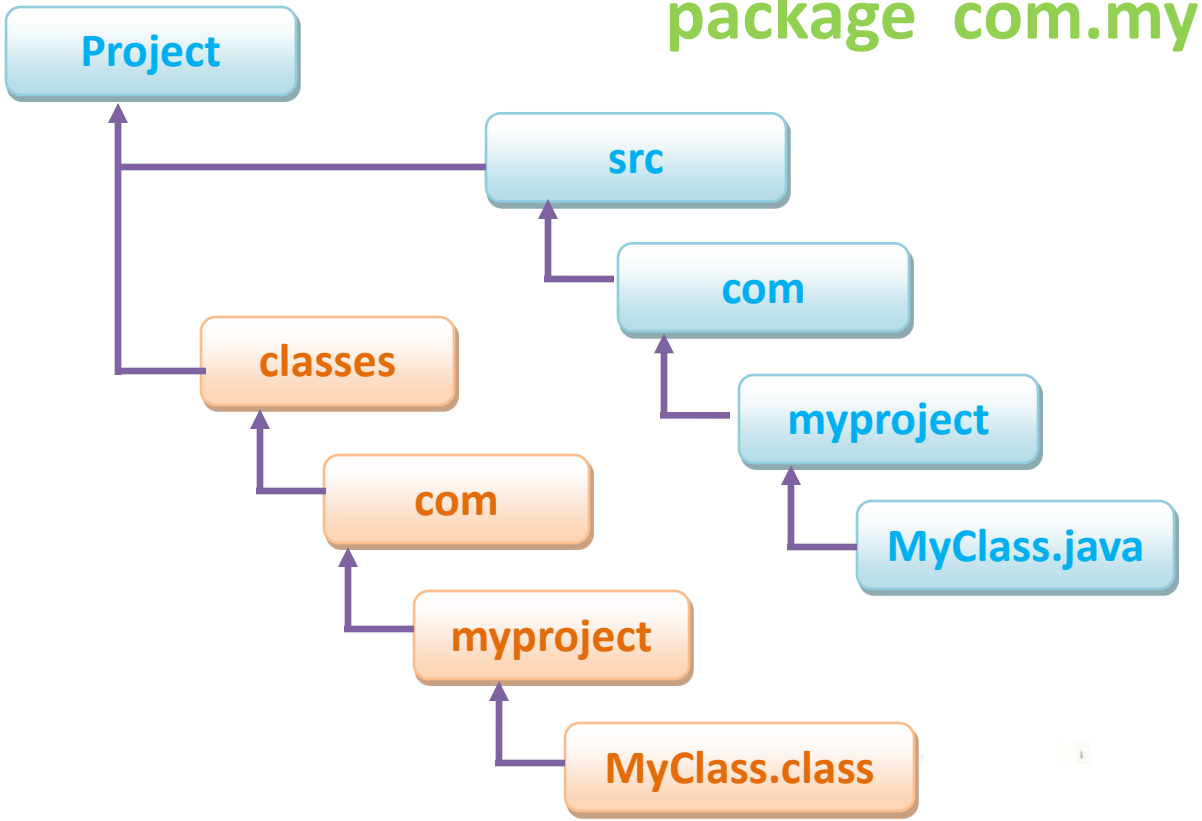
In this event, the suggested convention is to add an **underscore**

Legalizing Package Names	
Domain Name	Package Name Prefix
hyphenated-name.example.org	org.example.hyphenated_name
example.int	int_.example
123name.example.com	com.example._123name

Package

edureka!

```
package com.myproject;
```



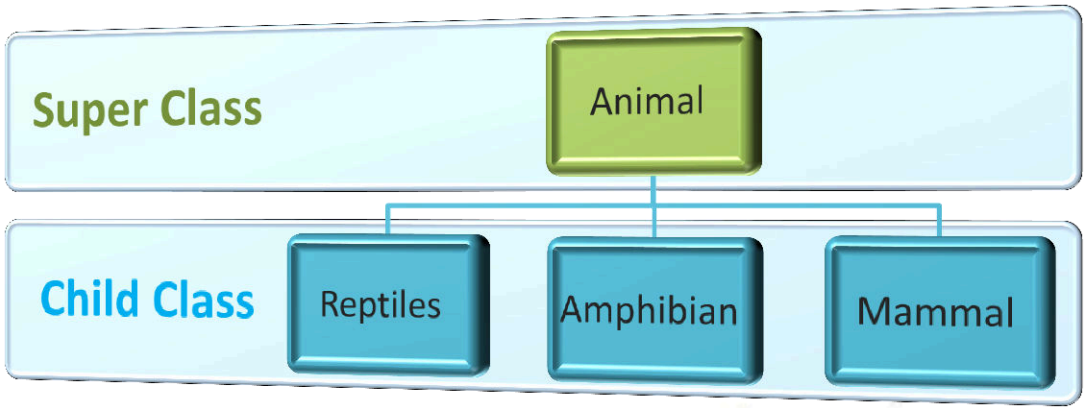
edureka!

Inheritance

Inheritance

edureka!

- The child classes inherits all the attributes of the parent class
- They also have their distinctive attributes



```
Class Animal {
//  Type : Not Human
//  Lives : On Land
//  Gives Birth :
}
```

```
Class Aquatic  extends Animal {
//  Lives : In Water
//  Gives Birth :
}
```

```
package com.edureka.animal;

public class Animal {

public String Type = "Not Human";
public String Lives = "In Water";

public void fun(){

    }

}
```

```
package com.edureka.animal;

public class Aquatic extends Animal{

String Lives = "In Water";

public void fun(){
            System.out.println("defined here");
      }
}
```

```
package com.edureka.animal;

public class Main {

public static void main(String[] args) {

        Aquatic ob = new Aquatic();

        System.out.println(ob.Type);
        System.out.println(ob.Lives);

         ob.fun();
    }
}
```

Super keyword

edureka!

- **super** is a keyword used to refer to the variable or method or constructor of the immediate parent class.

edureka!

Method Overloading

Method Overloading

edureka!

- The overloaded function must differ either by the number of **arguments** or **operands** or **data types**.
- The **same function name** is used for various instances of function call

Method Overloading

edureka!

```
package com.edureka.animal;

public class Animal {

public void fun(){
    System.out.println("Without Parameters");

    }

}
```

```
package com.edureka.animal;

public class Aquatic extends Animal{

public void fun(int num){
      System.out.println ("The number
                              passed is : " + num);
    }
}
```

Method Overloading

edureka!

```
package com.edureka.animal;

public class Main {

public static void main(String[] args) {

        Aquatic ob = new Aquatic();

 //without Parameters, defined in class Animal
        ob.fun();
//with Parameter, overloaded function in class Aquatic
        ob.fun(10);
    }
}
```

edureka!

Method Overriding

Method Overriding

edureka!

- The implementation in the subclass overrides (replaces) the implementation in the superclass
- It is done by providing a method that has same

1. **name**, same
2. **parameters**, and same
3. **return type** as the method in the parent class

Method Overriding

edureka!

```
package com.edureka.animal;

public class Animal {

int  fun(int a, int b){
     int c =  a + b;
     return c;
    }

}
```

```
package com.edureka.animal;

public class Aquatic extends Animal{

int  fun(int a, int b){
      System.out.println ("Sum by super class: " +
                                          super.fun(a, b));
       int c = a * b;
       return c;
      }
}
```

Method Overriding

edureka!

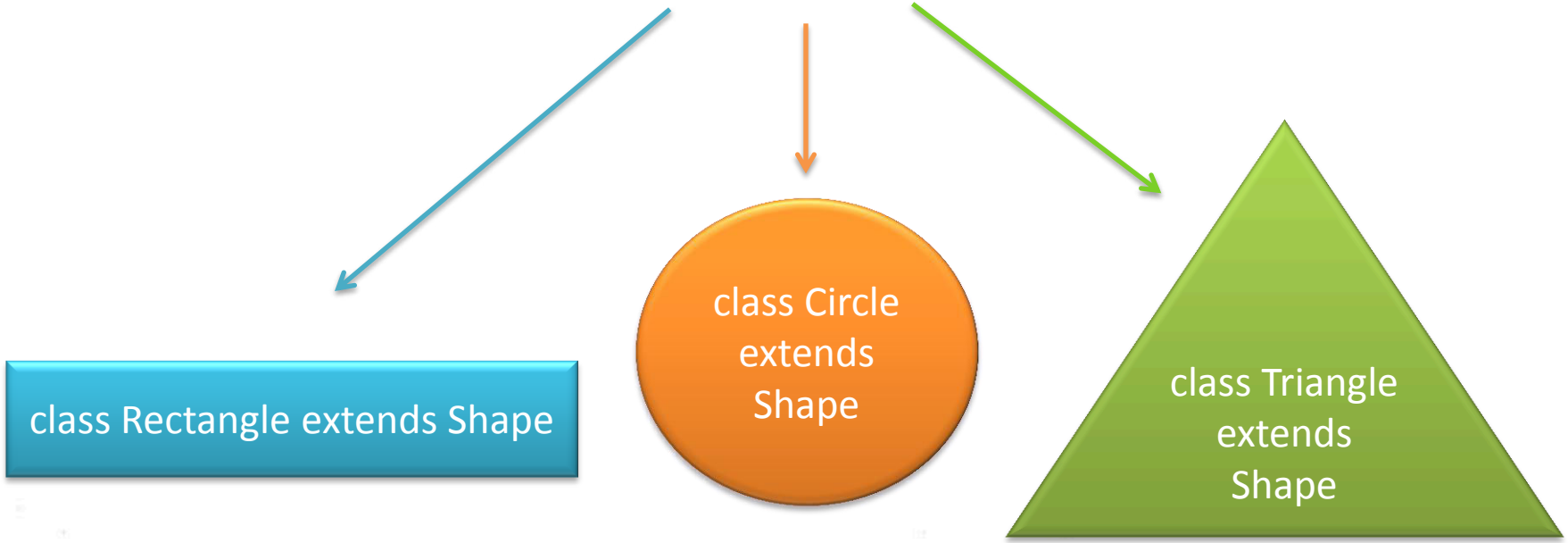
```
package com.edureka.animal;

public class Main {

public static void main(String[] args) {

        Aquatic ob = new Aquatic();
        System.out.println("Product by derived class : "+ ob.fun(2,3));

    }
}
```

edureka!

Abstract Class

Abstract Class

edureka!



What is an Abstract Class?

edureka!

- A class that is declared **abstract**
  - **Ex - abstract class Demo**
  - **It may or may not use abstract methods**

What is an Abstract Method ?

edureka!

- A method that is declared **abstract**
- A method that is declared **without an implementation**
  - Ex - **abstract void add(int x, int y);**

Example

```
public abstract class Shape {

//Abstract method i.e no implementation
abstract void Area();
}
```

```
public class Rectangle extends Shape {

@Override
void Area() {
Double area = length * width;

}
}
```

```
public class Circle extends Shape {

@Override
void Area() {
Double area =  3.14 * radius*radius;
}
}
```

eureka!

edureka!

Interface

Interface

edureka!

- Interfaces are declared using the **interface** keyword
- Interface can contain :
  - » **method signature**

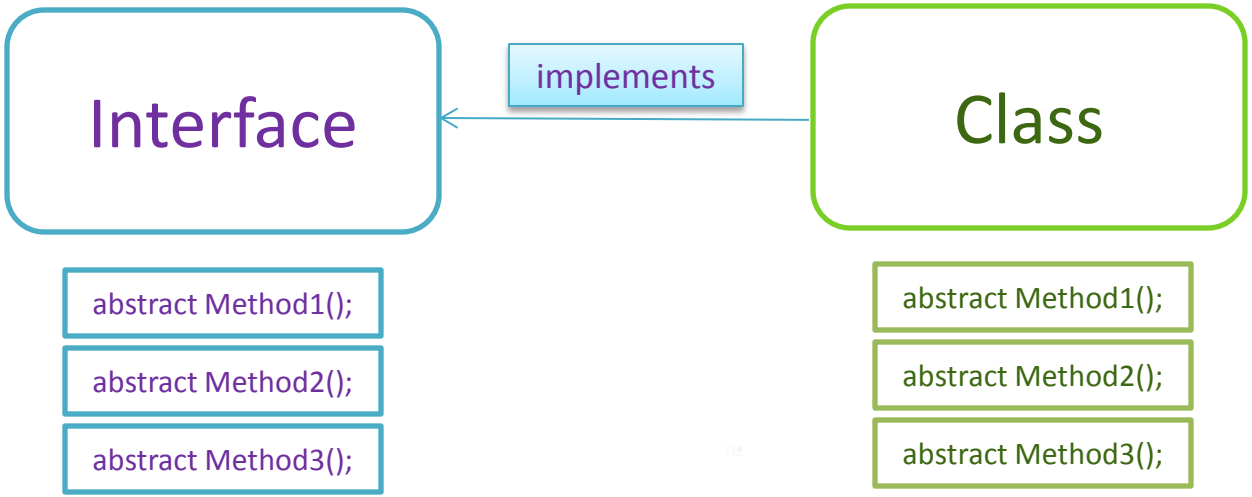
    (no implementation)
  - » **constant declarations**

    (variable declarations that are declared to be both **static** and **final**)

Interface

edureka!

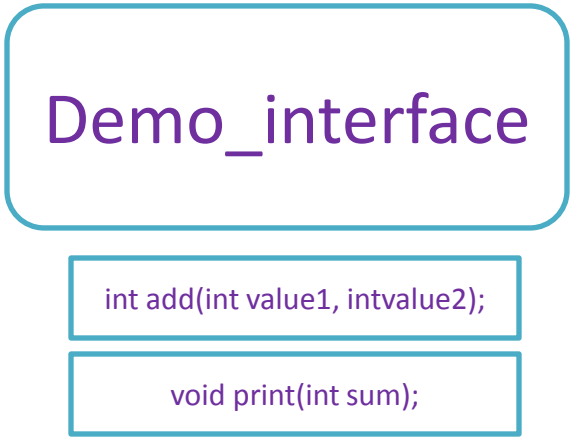
- A class that **implements an interface** must implement all of the methods described in the interface



How to create an Interface

edureka!

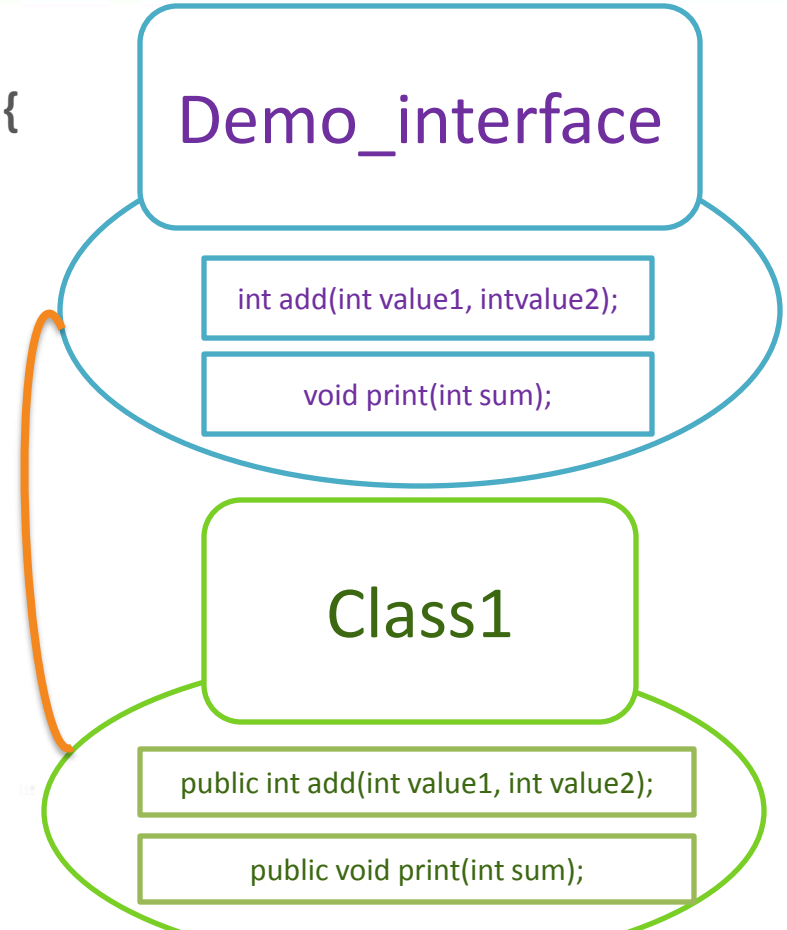
```
public interface Demo_interface {

int add(int value1, int value2);
void print(int sum);

}
```



How to implement the Interface

edureka!

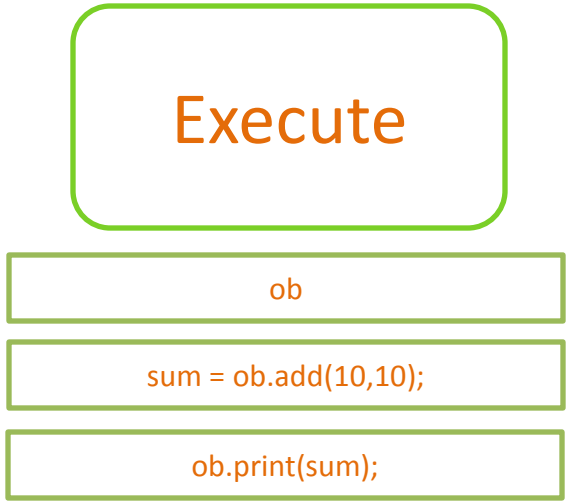
```
public class Class1 implements Demo_interface {
@Override
public int add(int value1, int value2) {
int sum = value1 + value2;
return sum;
}
@Override
public void print(int sum) {
System.out.println("The sum is : " + sum);
}
}
```



Using it in the main class

edureka!

```
public class Execute {

public static void main(String[] args) {

Class1 ob = new Class1();

int sum = ob.add(10, 10);

ob.print(sum);
}
}
```



edureka!

Input And Output

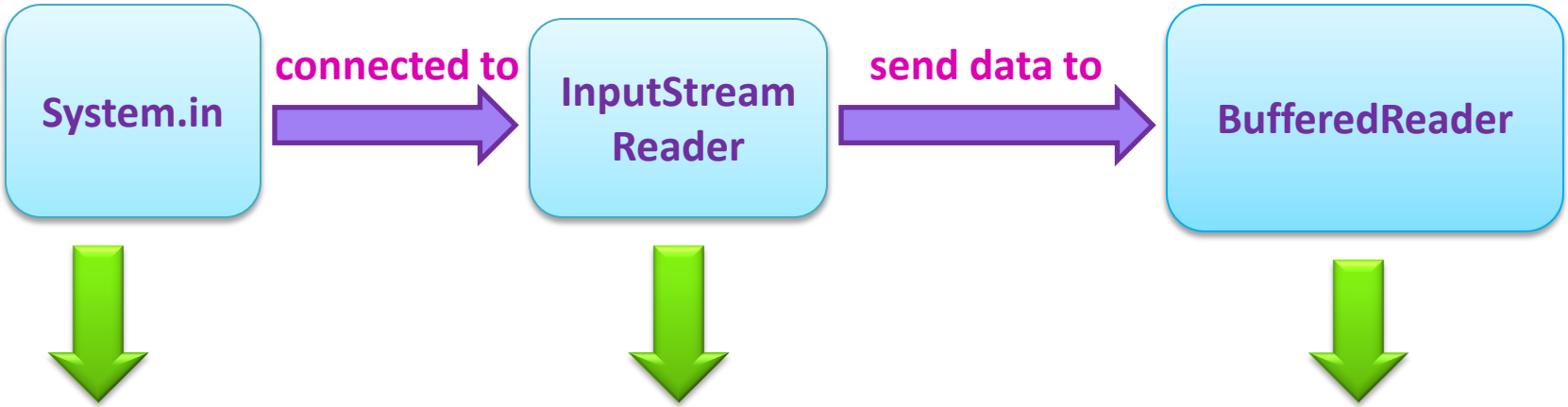
Input and Output

edureka!

- **Input** is the data given by the user to the program.
- **Output** is the data what we receive from the program in the form of result.
- **Stream** represents flow of data i.e. sequence of data.
- To give input we use **InputStream** and to receive output we use **OutputStream**.

How input is read from Keyboard?

edureka!



Reading Input from console

edureka!

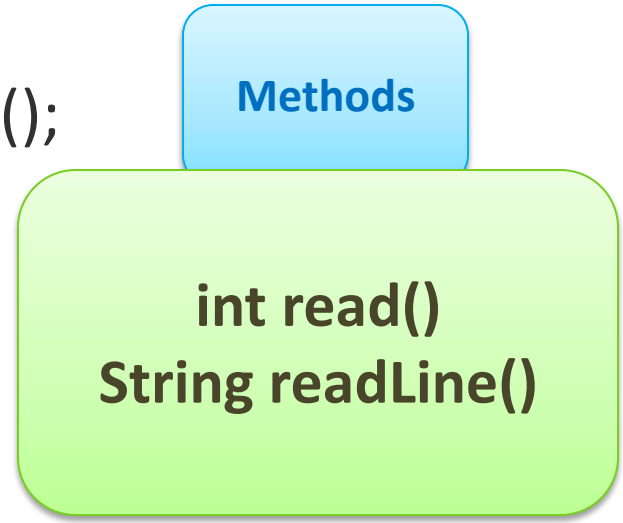
- **Input** can be given either from file or keyboard.

- Input can be read from console in 3 ways.
  - **BufferedReader**
  - **StringTokenizer**
  - **Scanner**

BufferedReader

edureka!

```
BufferedReader bufferedreader = new
        BufferedReader(new InputStreamReader(System.in));

int age = bufferedreader.read();
String name = bufferedreader.readLine();
```



- It can be used to accept **multiple inputs** from console **in single line** where as BufferedReader accepts only one input from a line.
- It uses **delimiter**(space, comma) to make the input into tokens.

StringTokenizer

edureka!

```
BufferedReader bufferedreader = new BufferedReader(new
                                    InputStreamReader(System.in));
        String input = bufferedreader.readLine();

StringTokenizer tokenizer = new StringTokenizer(input, ",");
String name = tokenizer.nextToken();
int age=Integer.parseInt(tokenizer.nextToken());
```

Scanner

edureka!

- It accepts multiple inputs from file or keyboard and divides into tokens.
- It has methods to different types of input(int, float, string, long, double, byte) where tokenizer does not have.

```
Scanner scanner = new Scanner(System.in);
        int rollno = scanner.nextInt();
        String name = scanner.next();
```

Writing output to console

edureka!

- The output can be written to console in 2 ways:

➢ **print(String)**-

```
System.out.print("hello");
```

➢ **write(int)**-

```
int input='i';
System.out.write(input);
System.out.write('/n');
```

- Q& A..?

Thanks..!