

# Object-Oriented Programming: Introduction to Java

Yi-ting Chiang

Department of Applied Mathematics  
Chung Yuan Christian University

November 14, 2023

# Programming Languages

Programming languages can be divided into three types:

- Machine languages
- Assembly languages
- High-level languages

Python, C, and Java are high-level programming languages. One of the differences between Python, C, and Java is the way they are executed:

- Python is a script language that its statements are written in a text file and can be executed without being compiled.
- C source codes have to be compiled into machine languages of the specific platform (OS) they are executed.
- Java programs is compiled into Java bytecode, which can be put into any JVM (Java Virtual Machine) in different platform to be executed.

Python and Java bytecode programs are portable. However, comparing to C programs, their performance is relatively poor.

# Write A Java Program

To write a Java program, you have to first declare a class. The syntax to declare a Java class is:

```
class JavaClassExample
{

}
```

Notice that

- Declare a class using the keyword “class” followed by the name of your class
  - The name of the class is its identifier
  - The name of an identifier cannot begin with a digit (0 to 9) and cannot contain spaces
  - Java is case sensitive. That is, uppercase chars (e.g. 'A') and lowercase chars (e.g. 'a') are different
- The left brace “{” denotes the start of the class declaration
- The corresponding right brace “}” denotes the end of the class declaration

# A Simple Java Program

```
public class MyFirstJava
{
    public static void main(String args[])
    {
        System.out.print("My Java program!");
    }
}
```

This example gives a simple Java program. In a Java program:

- The start point of a Java program is in the main method.
  - The prototype of main method is “public static void main(String[])”
- A public class must be written in a Java file with the same name as the name of this class.
  - There can only have one public class in a Java source file.

# A Simple Java Program

```
public class MyFirstJava
{
    public static void main(String args[])
    {
        System.out.print("My Java program!");
    }
}
```

More about the main method:

- The parentheses “()” followed by “main” denotes “main” is a method
- Any Java program must have one main method, which is where this program starts
- “String[] arg” is used to get the input arguments (as Java Strings) from the command line
- “void” means this method will not return anything
- The modifiers “public” and “static” will be explained later

# Comments in Java Program

Using comments in your Java source codes helps you and other people to understand your code.

```
/*
    Adding comments into your program.
    This program declare a public class with the name "MyFirstJava".
    We define one "method" main() in this class.
*/
public class MyFirstJava
{
    // Java program starts from the main() method.
    public static void main(String args [])
    {
        //This statement prints a message to the screen.
        System.out.print("My Java program!");
    }
}
```

You have to define classes and write the statements in Java. To declare a class in Java, use the keyword “class” followed by the name of your class.

# Compile A Java Program

A general procedure to develop a Java program from the command line is:

- ① Download and install Java Development Kit (JDK) into your computer if you don't have it
- ② Use any text editor to write your Java program and save it with file extension “.java”
- ③ Use Java compiler (generally “javac”) to compile your .java files
  - If there is no syntax error, bytecode files (with “.class” file extension) will be generated
- ④ To execute the .class file, use the command “java” to invoke JVM and load the java bytecode files.

You can also use IDEs (integrated development environments) to write your Java program. Here are some popular IDEs support Java:

- Eclipse
- NetBeans
- VS Code (Visual Studio Code)



# A Simple Introduction to Java

In the following, we give a simple introduction to Java. The topics are:

- Output messages to the screen
- Data types
- Operators
- Get input from keyboard
- Control statements
  - If-else, conditional operator, switch-case
  - while and for loop
- Define and call other methods

# Output Messages in Java Program

The example above shows how to print a string in Java. There is another method to do this. See the following example:

```
public class PrintLine
{
    public static void main(String args[])
    {
        System.out.print(" Print a string! ");
        System.out.print(" Print another string!");
    }
}
```

The output of this example is

```
Print a string! Print another string!
```

Notice that the two strings are printed in the same line. Method `print()` doesn't add any newline after the string.

# Output Messages in Java Program

You can use method `println()` to print a string. This method automatically adds a new line after this string. See the following example:

```
public class PrintTwoLines
{
    public static void main(String args[])
    {
        System.out.println(" Print a string!");
        System.out.print(" Print another with newline!\n");
    }
}
```

The output of this example is

```
Print a string!
Print another with newline!
```

Notice the `'\n'` in the end of the second string. Like C programming language, `'\n'` is the newline char. As a result, the second string on the screen also has a newline after it.

# Output Messages in Java Program

Recall the “printf” function in C language. There is a similar method available in Java. See the following example:

```
public class PrintfTwoLines
{
    public static void main(String args[])
    {
        System.out.printf("%s%n%s%n", "Print a string!",
                           "Print another string!");
    }
}
```

The output of this example is

```
Print a string!
Print another string!
```

“%s” means the data to be printed is a string. Both “\n” and “%n” in System.out.printf() are newline chars.<sup>1</sup>

---

<sup>1</sup>Note that System.out.print() and System.out.println() cannot use %n.

# Primitive Data Types

The following table gives the primitive data types in Java:

Data Type	name	memory size	comment
Integer	byte	1 byte	unsigned values
	char	2 bytes	
	short	2 bytes	
	int	4 bytes	
	long	8 bytes	
Floating point	float	4 bytes	
	double	8 bytes	
Boolean	boolean	undefined	VM dependent

# Primitive Data Types

The syntax of Java is very like that of C programming language. For example, the syntax to declare variables is:

```
<data_type> <variable_name> [= <init value>];
```

The following program shows how to declare and print variables:

```
public class DataTypeExample {  
    public static void main(String args[]) {  
        int i=10;  
        double d = 3.14;  
        float f = 3.14f; // the 'f' indicates that this 3.14 is float, not double  
        System.out.println(i);  
        System.out.printf("d=%f, f=%f.%n",d,f);  
    }  
}
```

The output is

```
10  
d=3.140000, f=3.140000.
```

Notice the different ways of printing variables demonstrated in this example. Also note that writing “float f=3.14;” is illegal because the data type of “3.14” is double.

# Reference Data Types

The data types that are not primitive in Java are reference data type. Reference data types are used to store the address of objects in computer's memory. So this kind of variables can refer to objects.

The most common Java reference data type is string. The following example shows how to declare and output Java strings:

```
public class StringExample
{
    public static void main(String args[])
    {
        String s1 = "A Java string\n";
        String s2 = "Another Java string";
        System.out.print("The first string: "+s1);
        System.out.printf("The second string: %s%n", s2);
    }
}
```

The output of this example is

```
The first string: A Java string
The second string: Another Java string
```

Actually, a string in Java is a kind of Java object. More details about Java object will be illustrated later.

# Array

You can also put several data with the same data type into an array. The syntax of array in Java is different with that in C. See the following example:

```
class ArrayExample {
    public static void main(String[] arg) {
        int[] a={1,2,3,4,5};
        String[] b={"one","two","three","four","five"};
        System.out.printf("%d: %s\n",a[0],b[0]);
        System.out.printf("%d: %s\n",a[4],b[4]);
    }
}
```

The output if this example is:

```
1: one
5: five
```

Note that the way to declare an array is different. In addition, the same with C language, the index of a length- $n$  array is between 0 and  $n - 1$ .



The following example demonstrates how to declare an array with a given length:

```
class ArrayLenExample {  
    public static void main(String[] arg) {  
        int[] a=new int[5];  
        int len=a.length;  
        System.out.printf("the length of array a is %d\n",len);  
    }  
}
```

Notice that we use “a.length” to get the length of array a. In Java, you can easily get the length of an array in this way.

The output if this example is:

```
the length of array a is 5
```

Similarly, we can also use multi-dimensional arrays. Here is an example:

```
class MultiDimArrayExample {  
    public static void main(String[] arg) {  
        int [][] a={{1,2},{3,4}};  
        System.out.printf("%d %d%n",a[0][0],a[0][1]);  
        System.out.println(a[1][0]+" "+a[1][1]);  
    }  
}
```

The output if this example is:

1	2
3	4

# Operators

Computations can be done by using operators. For example, you can use operators to assign values and perform arithmetic and boolean operations.

The table below lists some Java operators. Most of these are the same as C programming language:

Operator	Symbol	Description
Assignment	=	
Arithmetic	+, -, *, /	Four arithmetic operations
	%	Remainder
	++, -	Increase/Decrease by 1
Compound Assignment	+=, -=, *=, /=, %=	
Comparison	==, !=, >, <, >=, <=	
Logic	&&,	Logical AND and OR
	!	Logical Complement

# Operators

This example shows the result of performing the four arithmetic operations with integers in Java.

```
public class IntArithmeticExample
{
    public static void main(String args[])
    {
        int i=5, j=2;
        System.out.println(" Addition: " + (i+j)); // parentheses are necessary
        System.out.println(" Subtraction: " + (i-j));
        System.out.println(" Multiplication: " + (i*j));
        System.out.println(" Division: " + (i/j)); // the result is also an integer
    }
}
```

The output is:

```
Addition: 7
Subtraction: 3
Multiplication: 10
Division: 2
```

Notice that if the parentheses of  $(i + j)$  are omitted, the result will be "52", not 7. Moreover, performing division of two integers will also get an integer.

# Operators

The codes is an example of using boolean variables and the logical operators:

```
public class BooleanExample
{
    public static void main(String args[])
    {
        boolean is_raining=true, has_umbrella=false;
        System.out.println("It is raining: "+ is_raining);
        System.out.println("John has umbrella: "+ has_umbrella);
        System.out.println("John will get wet: "+ ((is_raining)
                                                    && (!has_umbrella)));
    }
}
```

The output is:

```
It is raining: true
John has umbrella: false
John will get wet: true
```

Notice that in the last println, we use parentheses to denote which operation should be performed first. Like C programming language, there is also operator precedence in Java. Use parentheses or check the document if you are uncertain of it.

# Get Keyboard Input

In C language, we often use `scanf()` to read input. In the following we give an example of getting user's input from the keyboard:

```
import java.util.*;

public class ReadFromKeyboard {
    public static void main(String[] arg)
    {
        float price = 0.0f;
        int num=0;
        String name;
        Scanner sin=new Scanner(System.in);

        System.out.print("Item name: ");
        name=sin.next();
        System.out.print("Price per item: ");
        price=sin.nextFloat();
        System.out.print("Number of items: ");
        num=sin.nextInt();
        price*=num;
        System.out.printf("%d %s costs %f dollars%n",
                           num, name, price);
    }
}
```

Input apple, 2.3 and 5, the output is:

```
Item name: apple
Price per item: 2.3
Number of items: 5
5 apple costs 11.500000 dollars
```

Scanner is one of the classes we can use to read the data from keyboard. We use "new" to create an instance of Scanner, "sin". Then we can use sin to read the data from the keyboard.

# Control Structure: If

Like C programming language, you can also use “if-else”, “switch-case”, and conditional operator<sup>2</sup> to choose the statements you want to execute or skip. Here is an example of if-else statement in Java:

```
import java.util.Scanner;
public class Compare {
    public static void main(String[] arg)
    {
        Scanner s = new Scanner(System.in);
        float i=s.nextFloat();
        float j=s.nextFloat();

        if(i*j > 0) System.out.printf("%f", i*j);
        else System.out.printf("%f", -i*j);
    }
}
```

This example gets two values from keyboard and compute the absolute value of their multiplication. Give two input values 4 and 2 or -4 and 2 from keyboard, both outputs are

8.000000

---

<sup>2</sup>(condition) ? expression1 : expression2

# Control Structure: If

The following example checks if a given year is leap year:

```
import java.util.*;
class LeapYear {
    public static void main(String[] arg) {
        Scanner sin = new Scanner(System.in);
        int year=sin.nextInt();
        boolean is_LeapYear;
        if(year % 400 == 0 )
            is_LeapYear=true;
        else if (year % 4 == 0 && year % 100 != 0)
            is_LeapYear=true;
        else is_LeapYear=false;
        System.out.printf("%d is leap year? %B%n",year,is_LeapYear);
    }
}
```

Input 1900, the result is:

```
1900
1900 is leap year? FALSE
```



# Control Structure: Conditional Operator

This examples shows the usage of conditional operator in Java.

```
import java.util.*;

class conditionalOP {
    public static void main(String[] arg) {
        Scanner sin = new Scanner(System.in);
        int value=sin.nextInt();
        int output = value > 0 ? value : -value;
        System.out.printf("Input value is %s%n",
            ((value>0) ? "positive" : "negative"));
        System.out.printf("|%d|=%d%n",value,output);
    }
}
```

If input -3, the output is:

```
Input value is negative
|-3|=3
```

If input 5, the output is:

```
Input value is positive
|5|=5
```

# Control Structure: Switch-Case

Java also has switch-case statement. However, there is some difference in Java switch-case statement. See the following example:

```
import java.util.*;
class SwitchCaseByString {
    public static void main(String[] arg) {
        Scanner sin = new Scanner(System.in);
        String degree=sin.next();
        switch (degree) {
            case "A": case "a":
                System.out.println("Excellent!");
                break;
            case "B": case "b":
                System.out.println("Very Good!");
                break;
            case "C": case "c":
                System.out.println("Good.");
                break;
            case "D": case "d":
                System.out.println("Not good!");
                break;
            case "E": case "e":
                System.out.println("Poor!");
                break;
            case "F": case "f":
                System.out.println("Fail!");
                break;
            default:
                System.out.println("Error input!");
        }
    }
}
```

Recall that in C language, you can only use integer in switch and constant integer followed by case. In Java, in addition to integer, you can use Java String in switch.

This example shows how to give different outputs according to the input string.

# Control Structure: Loop Statement

Most programming languages provide the mechanism called “loop” to repeatedly perform statements under some conditions. Java has four kinds of loop statements:

- while
- for
- for-each
- do-while


# Control Structure: While

The syntax of while in Java is the same as that in C language:

```
import java.util.*;
class WhileExample {
    public static void main(String[] arg) {
        Scanner sin = new Scanner(System.in);
        double sum = 0;
        while(sin.hasNext()) //press ctrl-z to end input
            sum += sin.nextDouble();
        System.out.printf("Sum of the input numbers=%f%n",sum);
    }
}
```

This example shows how to use while and the method “hasNext()” defined in Scanner to read values from keyboard and compute their summation. Note that as the function scanf in C language, you need to press ctrl-Z<sup>3</sup> to end the input.

---

<sup>3</sup>Press ctrl-Z under MS Windows. But press ctrl-D if you are under linux. 

# Control Structure: Do while

Unlike while, that the condition is checked first. In do-while, the statements in the loop will be executed at least one time. See the following example:

```
import java.util.*;
public class DoWhileExample {
    public static void main(String args[]) {
        int i = 1;
        int sum = 0;
        Scanner sin = new Scanner(System.in);
        int n = sin.nextInt();
        do {
            sum += i;
            i++;
        } while (i <= n);
        System.out.printf("Sum = %d\n", sum);
    }
}
```

This example reads a integer  $n$ , and computes  $1 + 2 + \dots + n$ . However, even the input value  $n < 1$ , the output will still be 1.

# Control Structure: For

For loop in Java also has the same syntax as that in C language. The syntax is:

```
for(①initialization;②condition;③adjustment)  
    ④statement;
```

Recall that the order of executing the four parts of for loop is

①②④③②④③②④③...

That is, the first part (①initialization) only execute one time.

# Control Structure: For

This example uses for loop to print the content of two arrays:

```
class ForExample {  
    public static void main(String[] arg) {  
        int[] a={1,2,3,4,5};  
        String[] b={"one","two","three","four","five"};  
        for(int i=0;i<b.length;i++)  
            System.out.printf("%d: %s%n",a[i],b[i]);  
    }  
}
```

Note that `b.length` is used to get the length of this array of string. The output of this example is:

```
1: one  
2: two  
3: three  
4: four  
5: five
```

# Control Structure: For

The example below use for loop to shows the difference between the arrays in C language and Java:

```
public class ForExample_Err {
    public static void main(String args[]) {
        int [][] M = {{1,2,3},{4,5}};
        for(int i=0;i<2;i++) {
            for(int j=0;j<3;j++) {
                System.out.printf("%d ",M[i][j]);
            }
            System.out.println("");
        }
    }
}
```

Running this example gets the following message:

```
1 2 3
4 5 Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 2
out of bounds for length 2
    at ForExample_Err.main(ForExample_Err.java:6)
```

This error (Exception) occurs when the program tries to output  $M[1][3]$ . The reason is that the length of  $M[1]$  is NOT 3, but 2. Recall that declaring an two-dimensional array in C language in this way gets an  $2 \times 3$  array. This is different in Java.



# Control Structure: For Each

Java provides a way to iteratively take an element from a collection. The syntax is:

```
for(<data type> <var_name> : <collection>)  
    statement;
```

Here <data type> is the data type of the element in the collection; <var\_name> is the variable name you define. <collection> gives from which the collection you want to get. For example, if the collection is an array, then the for each loop will get all the elements from this array.

# Control Structure: For Each

Let's see this for-each example:

```
public class ForEachExample_1 {  
    public static void main(String args[]) {  
        int[] M = {1,2,3};  
        for(int m: M)  
            System.out.printf("%d ",m);  
    }  
}
```

The output is:

1 2 3

The for each loop gets elements from M one by one. Each element in M is an integer, so we use int as the data type of m, which is a new-defined variable in the loop.

# Control Structure: For Each

Here is a more complex example:

```
public class ForEachExample_2 {  
    public static void main(String args[]) {  
        int [][] M = {{1,2},{3,4,5}};  
        for(int [] m: M) {  
            System.out.printf("The length of current m=%d%n",m.length);  
            for(int v: m) {  
                System.out.printf("%d ",v);  
            }  
            System.out.println("");  
        }  
    }  
}
```

The output of this example is:

```
the Length of current m=2  
1 2  
the Length of current m=3  
3 4 5
```

The example verifies that `M[0]` and `M[1]` have different length. Check what's the difference inside the parentheses of the two for each loops.

# Control Structure: Break

In Java, you can also use break or continue in a loop. Here is an example:

```
import java.util.*;
public class BreakExample {
    public static void main(String[] arg) {
        Scanner sin = new Scanner(System.in);
        int sum=0;
        while(sin.hasNext()) {
            int n = sin.nextInt();
            if(n<0) break;
            sum+=n;
        }
        System.out.printf("Sum of the positive numbers = %d%n",sum);
    }
}
```

This example computes the sum of the input values until a negative number is inputted or all the inputs have been read. If you input the following values in the same line:

```
1 2 3 -4 5
```

Then the output is:

```
Sum of the positive numbers = 6
```

Because when -4 is read, the loop stops by the break statement. The value 5 is not be used.

# Control Structure: Continue

Statement continue will not end the loop. It only skips all the statements that have not been executed in current iteration and starts the next iteration. See the following example:

```
import java.util.*;
public class continueExample {
    public static void main(String[] arg) {
        Scanner sin = new Scanner(System.in);
        int sum=0;
        while(sin.hasNext()) {
            int n = sin.nextInt();
            if(n<0) continue;
            sum+=n;
        }
        System.out.printf("Sum of the positive numbers = %d%n",sum);
    }
}
```

Give the following input in the same line and press ctrl-Z (Use ctrl-D under linux):

1 2 3 -4 5

Then the output is:

Sum of the positive numbers = 11

This time, only -4 is skipped.

# Control Structure: Define and Call Methods

Recall that you can define your own functions in C programming language. In Java, we call the “functions” inside the class “methods”. Methods (functions) cannot be defined outside a Java class. Let’s see the example:

```
import java.util.*;
class LeapYear_2 {
    private static boolean isLeapYear(int year) {
        if(year % 400 == 0 ) return true;
        else if (year % 4 == 0 && year % 100 != 0)
            return true;

        return false;
    }
    public static void main(String[] arg) {
        Scanner sin = new Scanner(System.in);
        int year=sin.nextInt();
        System.out.printf("%d is leap year? %b%n",year,isLeapYear(year));
    }
}
```

In this example, we define a method “isLeapYear” to decide if a given year is leap year or not. Notice the prototype of this method:

```
private static boolean isLeapYear(int year);
```

“boolean” means the output data type of this method. “int year” in the parentheses gives the name and the data type of the input parameters. The output is as the example we gave in Page 24<sup>4</sup>.

We will explain what is “private” and “static” later.

---

<sup>4</sup> “true” and “false” is in lower case here.

# Control Structure: Define and Call Methods

If there the method has no return value, put the keyword “void” before the method's name. However, if there is no input parameter, just put nothing inside the parentheses:

```
import java.util.*;
class LeapYear_3 {
    private static boolean isLeapYear(int year) {
        if(year % 400 == 0 )
            return true;
        else if (year % 4 == 0 && year % 100 != 0)
            return true;

        return false;
    }

    private static void print(String s) {
        System.out.println(s);
    }

    private static void chkLeapYear() {
        Scanner sin = new Scanner(System.in);
        int year=sin.nextInt();
        print(year+" is leap year? "+isLeapYear(year));
    }

    public static void main(String[] arg) {
        chkLeapYear();
    }
}
```

# Generate Random Numbers

Java provides several ways to generate random numbers. Here we introduce three of them:

- `Math.random()`
- `java.util.Random;`
- `java.security.SecureRandom;`



# Generate Random Numbers: Math.random()

The easiest way to get a random value in  $[0,1]$  is to use `Math.random()`. Here is an example:

```
class RandomDouble {  
    public static void main(String [] args) {  
        double d=Math.random();  
        System.out.println(d);  
    }  
}
```

The public static member function “`Math.random()`” provided by the class `java.util.Math` generates a double value, uniformly distributed in  $[0,1]$ <sup>5</sup>.

---

<sup>5</sup>There is no true random number generator today. All random number generators are pseudorandom.

# Generate Random Numbers: java.util.Random

You can use java.util.Random to get different types of random numbers. Here is an example:

```
import java.util.*; //To use java.util.Random;
class RandomNumbers {
    public static void main(String[] args) {
        Random rnd=new Random();
        System.out.printf(" Boolean: %b%n",rnd.nextBoolean());
        System.out.printf(" double: %f%n",rnd.nextDouble());
        System.out.printf(" float: %f%n",rnd.nextFloat());
        System.out.printf(" int: %d%n",rnd.nextInt());
        //get a value in [0,9]
        System.out.printf(" bounded int: %d%n",rnd.nextInt(10));
        System.out.printf(" long: %d%n",rnd.nextLong());
    }
}
```

In the example above:

- nextBoolean(): return an uniformly distributed boolean value.
- nextDouble(): return an uniformly distributed double value in [0.0,1.0).
- nextFloat(): return an uniformly distributed float value in [0.0,1.0).
- nextInt(): return an uniformly distributed int value. The result can be all  $2^{32}$  int values.
- nextInt(int b): return an uniformly distributed int value in [0,b).
- nextLong(): return an uniformly distributed long value. The random seed in java.util.Random is 48 bits, so some long values will not be generated.

# Generate Random Numbers: `java.security.SecureRandom`

`java.security.SecureRandom` class provides strong random number generator that can be used in cryptographic algorithms. The main difference between this class and `java.util.Random` is<sup>6</sup>:

- Random seed size: 128 bits. So to break the code,  $2^{128}$  attempts are required. In the case of `Random`, it requires only  $2^{48}$ .
- Seed generation: `Random` uses system clocks while `SecureRandom` takes random data from the states of the OS.
- Generating function: Use a different random number generator<sup>7</sup>.
- Security: It's recommended that Use `SecureRandom` to implement security-critical applications.

---

<sup>6</sup>From: <https://www.geeksforgeeks.org/random-vs-secure-random-numbers-java/>

<sup>7</sup>"SHA1PRNG"

# Generate Random Numbers: java.security.SecureRandom

The following example uses SecureRandom to generate different types of random numbers:

```
import java.security.SecureRandom;

class SecureRandomNumbers {
    public static void main(String[] args) {
        SecureRandom srnd = new SecureRandom();
        System.out.printf("double: %f%n", srnd.nextDouble());
        System.out.printf("float: %f%n", srnd.nextFloat());
        System.out.printf("int: %d%n", srnd.nextInt());
        //get a value in [0,9]
        System.out.printf("bounded int: %d%n", srnd.nextInt(10));
        System.out.printf("long: %d%n", srnd.nextLong());
    }
}
```

# Generate Random Numbers: Random Seed

Sometimes you may need to repeatedly generate the same sequence of random numbers. To do this, you can set a fix random seed:

```
import java.util.Random; //To use java.util.Random;
class RandomSeed {
    public static void main(String[] args) {
        Random rnd=new Random(10); //set random seed when init
        for(int i=0;i<2;i++)
            System.out.printf("Random value: %d%n",rnd.nextInt());

        rnd.setSeed(10); //set to the same seed
        for(int i=0;i<2;i++)
            System.out.printf("Random value: %d%n",rnd.nextInt());
    }
}
```

You can find that the result of the two sequences in the two for loop are the same:

```
Random value: -1157793070
Random value: 1913984760
Random value: -1157793070
Random value: 1913984760
```

# Generate Random Numbers: Random Seed

Here is a Java exmple that simulates rolling a die:

```
import java.util.*; //To use java.util.Random;

public class DieRolling {
    public static void main(String[] args) {
        Random rnd=new Random(10); //random seed = 10
        int[] Dies= new int[6]; //default value is zero
        Scanner sin = new Scanner(System.in);
        System.out.print("Number of random values: ");
        int N=sin.nextInt();
        for(int i=0;i<N;i++) {
            int v=rnd.nextInt(6); //get 0 to 5
            Dies[v]++;
        }
        sin.close(); //close input stream

        for(int i: Dies)
            System.out.println((double)i/N);
    }
}
```

This example first reads the number of times to roll the die. The number of each side facing upwards is recorded in the array "Dies". finally, the probability of the six sides facing upwards is computed and output.