

Example Program:

```
import java.util.Scanner; //Scanner lets us read input from the keyboard

//calculate the product of three integers given by the user
public class Product
{
    /*
     * This method reads three integers from the user and prints their product
     */

    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);

        int x; //first value given by user
        int y; //second value given by user
        int z; //third value given by user
        int result; //the product

        System.out.println( "Enter first integer: "); //prompt for input
        x = input.nextInt(); //read first integer from keyboard

        System.out.println( "Enter second integer: "); //prompt for input
        y = input.nextInt(); //read second integer from keyboard

        System.out.println( "Enter third integer: "); //prompt for input
        z = input.nextInt(); //read third integer from keyboard

        result = x*y*z; //calculate the product of the three input numbers

        System.out.printf("Product is %d\n", result); //print the result
    } //end main method
} //end class Product
```

a. Primitive Data Types (there are 8)

- i. int
 1. An integer (a whole number) like 12, -5, or 0
 2. 32 bit signed number
 3. Uses 2's complement to store it (see below for explanation)
 4. Range (due to only having 32 bits to store it) from -2,147,483,648 to 2,147,483,647
- ii. long
 1. an integer value but has a larger range than int
 2. You must write it as 7L for the number 7 as a long
 3. 64 bit signed number
 4. Uses 2's complement
 5. Range 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- iii. short
 1. an integer value with a smaller range and smaller to store so saves memory if needed
 2. 16 bit signed number
 3. Uses 2's complement
 4. Range -32,768 to 32,767
- iv. byte
 1. an integer value with a smaller range
 2. 8 bit signed number
 3. Uses 2's complement
 4. Range -128 to 127
- v. double
 1. A decimal number
 2. 64 bit IEEE double precision floating point
 - a. IEEE has positive and negative values, also a positive and negative 0, a positive and negative infinity, and a not a number (NaN) value used for example if you divide by 0 which is not allowed.
 3. Can be written 1.8
 4. Can be written in scientific notation
 - a. 1E4 is 1×10^4
 - b. 5.3E-3 is 5.3×10^{-3} or 0.0053
 5. In a real program you wouldn't use double or float for precise values, you need to use the BigDecimal class that Java provides.
- vi. float
 1. a decimal number, smaller range than doubles and saves memory if needed.
 2. 32 bit IEEE single precision floating point
 3. Must write it as 8.5f if you want the number 8.5 as a float
- vii. boolean
 1. Boolean notation is either true or false (sometimes use 0 and 1 in science)
 2. We type true and false in Java
- viii. char
 1. characters including a character version of numbers (1) but you can't do math with them, and letters
 2. 16 bit Unicode characters (see appendix A in your book for some of the Unicode and the characters they represent)
 3. Range '\u0000' (0) to '\uffff' (65535)

2's Complement:

-leading digit is the sign. A leading 0 is positive, a leading 1 is negative

-for 32 bit numbers we have these examples:

0000 0000 0000 0000 0000 0000 0000 0000 = zero

0000 0000 0000 0000 0000 0000 0000 0001 = one

0000 0000 0000 0000 0000 0000 0000 0010 = two

...

0111 1111 1111 1111 1111 1111 1111 1111 = 2,147,483,645 in base 10

1000 0000 0000 0000 0000 0000 0000 0000 = -2,147,483,648 in base 10

...

1111 1111 1111 1111 1111 1111 1111 1101 = - 3

1111 1111 1111 1111 1111 1111 1111 1110 = - 2

1111 1111 1111 1111 1111 1111 1111 1111 = - 1

Table on arithmetic

Operation	Operator in Java	Example in Math	Example in Java
Addition	+	f+7	f+7
Subtraction	-	p-c	p-c
Multiplication	*	bm	b*m
Division	/	x/y	x / y
Remainder (Modulo)	%	r mod s	r % s

Table on operators

Algebraic operator	Java operator	Java example	What it means
=	=	x ==y	x is equal to y
≠	!=	x != y	x is not equal to y
>	>	x>y	x is greater than y
<	<	x<y	x is less than y
≥	>=	x>=y	x is greater than or equal to y
≤	<=	x<=y	x is less than or equal to y

Strings

- Not a fundamental data type
- Definition: a sequence of characters (including letters, numbers, punctuation, spaces, etc)
- Length of the string is defined as the number of characters in it. After you declare a String variable you can get the length with name.length();
- You can use an empty string "" as a default, which has a length of 0.
- String is an object, so it must be capital when you declare String variablesString
 - String name = "Alicia";
- Concatenation
 - Two strings (either variables or literals) can be put together to form one string with the + sign.
 - This also works if you concatenate a string and a number, because the number is changed into a string and then they are concatenated.
 - This is used a lot with print statements
 - System.out.println("The result of this program is: " + result);

- g) Change Strings to an int or a double (only if the String literal is a number)
 - i. Example: `String number = 45; int nu = Integer.parseInt(number); // nu becomes 45`
 - ii. `Double.parseDouble()` also exists

h) Formatted output

- a. `System.out.printf("There are $%d in your account", balance);`
- b. `System.out.println(" * \n *** \n*****");`

```

      *
     ***
    *****
  
```

i) Methods of String class

- a. See Java API for complete listing of methods (or file posted on our webpage)
- b. To use a method of the String class on a specific string, you type `ObjectName.method()`;
- c. Strings are made up of a bunch of characters, you can pull out the *i*th character of a String with the `charAt()` method.
 - i. First, we need to note that we start our counting at position 0. (This will also be true of arrays when we learn about them). But when we count the length of the String it starts counting at 1. So this string has a length of 6, but the characters are labeled `char0` to `char5`. Think of this as, this word clearly has 6 letters, but we always start labeling at the 0th one.

A	l	i	c	i	a
0	1	2	3	4	5

- ii. `charAt()` is NOT a static method, because it has different results for each String
- iii. `String myName = "Alicia";`
`char myFirstLetter = myName.charAt(0);`
`///result of myFirstLetter is 'A'.`
- d. Pull out a part of the String using the `substring()` method
 - i. `substring()` has two parameters: the start point for the substring, and the char AFTER the endpoint for the substring. This is still counted from `char0`.
`String myName = "Alicia Maxwell";`
`String myFirstName = myName.substring(0,6);` (this gives "Alicia")
 - ii. There is also a version of `substring()` that has only one parameter: the start point for the substring (counting from `char0`). Then the entire rest of the string from the start point on is taken
`String myName = "Alicia Maxwell";`
`String myLastName = myName.substring(7);` (this gives "Maxwell")
- e. Compare Strings using the `equals()` method, not `==`
`String name = "Keladry";`
`String nickname = name.substring(0,3);`

```
if (name == "Keladry") //true because name was declared to be this String literal
```

```
if(nickname == "Kel")//although we can tell this is true, the
//computer cannot, because they are
//different objects
```

```
If (nickname.equals("Kel")); //true
```

- f. Compare Strings using the compareTo() method, not > or <
- i. lexicographic ordering
 1. This is basically dictionary ordering
 2. Capital letters come before lowercase (Z precedes a)
 3. Spaces come before any letter or symbol
 4. Numbers come before letters
 5. Punctuation has its own ordering which is in appendix A
 - ii. Example


```
String animal1 = "cat";
String animal2 = "zebra";

animal1.compareTo(animal2);
```
 - iii. This method returns an integer.
 1. If the result is <0, then string1 goes first.
 2. If result is >0, then string2 comes first.
 3. if result is 0 then they are equal.

Table of Escape Sequences

Escape Sequence	Description
\n	Newline. Position the screen cursor at the start of the next line
\t	Tab. Move the cursor to the next tab stop
\r	Carriage Return.
\\	Backslash. Print out the \ character
\"	Double quote. Print out the " character

Table of placeholders in the formatted output (ex: the last line in the example)

What you type	What kind of value you output
%s	String
%d	int
%f	double
%.2f	double with two decimal points like 3.45 or 12.34. Note that if you change 2 to a different number, you get that many digits after the decimal point
%c	char

Random Class

A. Constructing a Random Generator

- a. Pseudorandom numbers: drawn from a sequence of numbers that does not repeat for a very long time and so appears to be random.
- b. Zero argument constructor: `Random rng = new Random();`
- c. Seeding: `Random rng = new Random (long someSeedValue);`
 - i. Note that to use the number 1 as my seed, I need to type 1L so the computer knows it is of type long, not type int.
 - ii. The seed is used in the algorithm that computes the pseudorandom number. The zero argument constructor uses the timestamp from your computer at the time you run the program as the seed. Random number generators that have the same seed always generate the same random numbers in the same sequence. Thus if you know what seed you are giving it and which algorithm is being used, then you can always know exactly what pseudorandom number will be generated when

B. Methods of Random

- a. `nextInt(n)`
 1. Return a random int or floating point value between 0 (inclusive) and n(exclusive)
 2. Note that this is written $0 \leq x < n$. This exclusive at the top matters, and make sure you think twice about it in your code to make sure you are doing what you want and not off by one.
- ii. `nextInt()`
 1. Returns a random number in the range of allowed ints in Java (which is about -2.1billion to 2.1 billion)
- iii. `nextDouble()`
 1. Return a random double between 0 (inclusive) and 1 (exclusive)
- iv. There are other methods that we will not use in this class. Some of them are listed below.
 1. `nextGaussian()` returns a random double from a Gaussian distribution, which you need in statistics, physics, and other sciences
 2. `nextBoolean()` returns true or false with approximately equal probability
 3. `nextFloat()` returns a float between 0.0 and 1.0
 4. `nextLong()` returns one of the 2^{64} long values in the range of allowed longs

C. Shifting the values into the desired range

- a. Generate an int between 8 and 11
 - i. `nextInt(4)` will return a value between 0 and not-including 4
 - ii. to get the lower bound to change from 0 to 8, we just need to add 8 to 0, suggesting `nextInt(4) + 8`
 - iii. What will this do to the upper bound? It will turn from 4 (exclusive) to $4+8=12$ (exclusive) so the highest allowed integer will be 11, just as we want.
 - iv. Conclusion: use `int n = rng.nextInt(4) + 8;`
- b. Generate a number in the range of 1-6
 - i. `nextInt(6)` gives you a number $0 \leq x \leq 5$
 - ii. So if we just add 1 to that, it will move to being a range of $1 \leq x \leq 6$
 - iii. Solution: `int roll = rng.nextInt(6)+1;`
- c. Generate a floating point number between 0 and 2.
 - i. `rng.nextDouble()` returns a value between 0 and 1
 - ii. Then to change this to a value between 0 and 2, we can just multiply by 2
 - iii. Solution: `double n = 2*rng.nextDouble() ;`