# Introduction to Java

By Paul Baumgarten
pbaumgarten.com

July 2020 edition

# Introduction to Java programming

## 1. Hello world

For the first few weeks we will use an online service, https://repl.it/ to complete our programming. This will make it easy for me to monitor your progress in these early lessons. Once we have completed the introductory activities, we'll spend a lesson getting your personal laptop setup to run Java programs.

### Study your first program

Your first program looks like this...

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

There is a lot of technical grammar and syntax in this program, given this is supposed to be the "first and easiest" starting point! Take a close look at the different elements. What do you predict this program will do?

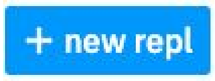
### Sign up to Repl.it

In your web browser, open https://repl.it and create a new account using your school Google Account.

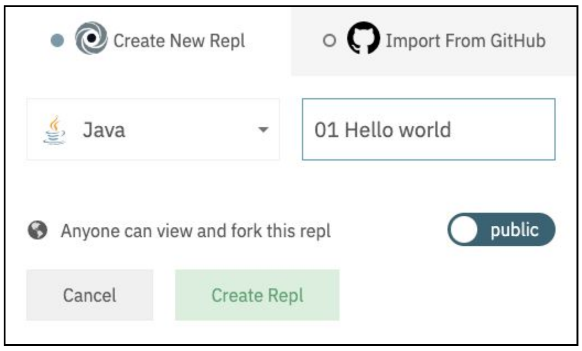
Respond to the Google Classroom question where I ask you to inform me of your repl.it username. This will allow me to monitor your progress.

### Create a Java program on Repl.it

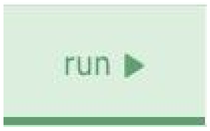
Find the "new repl" icon and click it.



In the new window that appears, select Java as your programming language and give your repl project a meaningful name. Repl will create random names that will be impossible for you to keep track of otherwise. In the example below I am suggesting to use a number indicating lesson number, followed by a brief name for the exercise.



You will notice your first program is actually the default that repl provides, so nothing to type for this first program. Click the run icon and see if it behaves the way you predicted.



### Modify

Experiment with modifying the sample program. Posit: how might you display multiple lines of messages?

Pair with someone else. Set your partner a multiline message you want them to create.

## Discussion

We will discuss the first program as a class, including some of the syntax and grammar involved. You can safely consider a lot of it as a "boilerplate" that is "just required" for now. Over the coming weeks you will learn what all the different elements do.

Some immediate things to draw your attention to:

- Java is case sensitive.
  `System.out.println` and `System.Out.PrintLn` are not the same thing!

- Syntax must be precise. Computers are exact tools and can only interpret what you provide according to set strict rules.

- Most instructions require a set of parenthesis (rounded brackets). Inside those parenthesis go the parameters. This is the information the instruction requires in order to run. If the instruction requires multiple parameters, the order of the parameters matters. You would separate each parameter with a comma.

- When we need to provide text information, we use "quotes" to indicate the start and end of the text.

- The curly braces `{ }` enclose "blocks" of code. They are used to group parts of your program together. Every opening brace requires a corresponding closing brace.

- Every instruction in Java terminates with a semicolon `;` which you can think of as the equivalent to a full stop.

- The indentation provided is not mandated by Java, but is mandatory by me. Just because this will run does not mean you should ever do it!

```
class Main{public static void main(String[] args){System.out.println("Hello
world!");}}
```

- Programmers use indentation to make code more readable. The general rule of thumb:
  - Every time you open a new block with `{` indent future lines of code.
  - Every time you close a block with `}` shift your indentation left again.
  - One instruction per line. Once you use a semicolon, move on to a new line.

- In-code comments are critical for readability. Comments are a way for you to add notes to your code for your future information. They are ignored by Java. There are two ways of adding comments to your code in Java.

```
class Main {
  public static void main(String[] args) {
    // This is a one line comment
    System.out.println("Hello world!");
  }
}
```

```
class Main {
  public static void main(String[] args) {
    /* This is a multiline comment
    It will keep going until I close it
    like this... */
    System.out.println("Hello world!");
  }
}
```

- You should also use comments to add links to tutorials or resources that helped you solve a particular issue as the example below shows. There are two benefits for this…
  - It helps you when you are looking at your code in the future to figure out why you did something a particular way.
  - It is the academically honest approach to give credit where credit is due. This will be important for your assessments later. Get in good habits now.

```
class Main {
  public static void main(String[] args) {
    // "Not a statement" error fixed via
    // https://stackoverflow.com/questions/36089158/java-hello-world
    System.out.println("Hello world!");
  }
}
```

---

## For future reference

- Sometimes when referring to Java tutorials online you need to know the version of the Java Runtime Engine (JRE) you are running as some features have only been introduced in newer versions. The following is a handy bit of reference code that will advise the version of Java you are running.

```
class Main {
    public static void main(String[] args) {
        System.out.println("Java version "+System.getProperty("java.version"));
    }
}
```

## 2. Variables & data types

Printing a few messages to the screen is all well and good but not terribly useful. In order to actually compute anything is going to require the use of variables.

A variable is a named memory location reserved for you to store values under that name. This will allow us to use the result from one calculation as the input for the next.

In addition to requiring a unique name, variables also require a *data type*. Because computers are far more capable than the humble calculator, they can store many different types of values into a variable, but for the programming language to respond correctly it must know what type of data is being stored in that variable.

There are a lot of data types available, and later on you will even create your own, but for now let's consider some of the most common ones.

| Java name | Description |
|---|---|
| boolean | A boolean is a type that can only have one of two possible values: **true** or **false**. |
| int | An integer number.<br><br>It has 32 bits of memory allocated (we'll get into that later). For now, the significance is that integers must be between the range of<br>−2'147'483`648 to +2'147'483'647. |
| long | A "long" integer number.<br><br>It has double the memory allocation, 64 bits. The number range permissible is<br>−9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 |
| float | Floating point number which means it is capable of storing decimals. This uses 32 bits with 1 bit for sign, 8 bit exponent, 23 bits significant figures. |
| double | "Double" precision floating point number meaning it has 64 bits of memory allocated. 1 bit sign, 11 bit exponent, 52 bit significant figures. Be aware that **double**, rather than **float**, is the default for most Java math functions. |
| String | A "string of characters", more easily known as "text". |

A variable is created by using the keyword of the data type, followed by a unique name for the variable, finally (optionally) you can assign it an initial value.

```java
class Main {
    public static void main(String[] args) {
        int num = 10; // Create integer variable num
        System.out.println(num);
    }
}
```

# 3. Numbers

## Declaring

The following code creates two integers, a and b. Predict the outputs and then test your understanding.

```java
class Main {
    public static void main(String[] args) {
        int a = 10;
        int b = a;
        a = 5;
        System.out.println(a);
        System.out.println(b);
    }
}
```

In like manner, declaring doubles looks very similar but the output will be slightly different. Can you predict the difference?

```java
class Main {
    public static void main(String[] args) {
        double a = 10;
        double b = a;
        a = 5;
        System.out.println(a);
        System.out.println(b);
    }
}
```

## Arithmetic operations

All the usual arithmetic operations are available for use in Java. Can you determine what all the symbols represent? Predict `f` and `g` in particular.

```
class Main {
    public static void main(String[] args) {
        int a, b, c, d, e, f, g;
        a = 17;
        b = 4;
        c = a + b;
        d = a - b;
        e = a * b;
        f = a / b;
        g = a % b;
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
        System.out.println(e);
        System.out.println(f);
        System.out.println(g);
    }
}
```

Try the above again, but first convert all the data types to float and note the slight differences.

## Input numbers

Our program would be far more useful if the user can enter their own numbers instead of needing them pre-entered into code. To have the program prompt for user input would look like this...

```java
// Import the Scanner object from the java.util library
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        int a, b, c, d, e, f, g;
        // Create a keyboard reading object
        Scanner keyboard = new Scanner(System.in);
        // Prompt the user & collect their responses
        System.out.print("Enter value for a:");
        a = keyboard.nextInt();
        System.out.print("Enter value for b:");
        b = keyboard.nextInt();
        // The rest is the same as before
        c = a + b;
        d = a - b;
        e = a * b;
        f = a / b;
        g = a % b;
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
        System.out.println(e);
        System.out.println(f);
        System.out.println(g);
    }
}
```

As you may infer, there is also a **`keyboard.nextDouble()`** function available if you need it.

## Math library

There are a whole range of useful functions in the Math library. Here are some of the more useful ones you are likely to use.

| | |
|---|---|
| `Math.pow( base, exponent )` | Perform exponential calculation, base raised to exponent. |
| `Math.sqrt( num )` | What is the square root of num? |
| `Math.round( num )` | Round num to the nearest integer value |
| `Math.floor( num )` | Truncate any decimals off of num down to the nearest integer value |
| `Math.abs( num )` | What is the absolute value of num? |
| `Math.random()` | Generate a random number between 0 and 1 |
| `Math.max( a, b )` | Returns the larger of a or b |
| `Math.min( a, b )` | Returns the smaller of a or b |
| `Math.PI` | The value of the mathematical constant PI, ie: 3.1415.... |
| `Math.E` | The value of the mathematical constant E, ie: 2.7183.... |
| `Math.sin( num )` | What is the sine value of num? (answers in radians) |
| `Math.cos( num )` | What is the cosine value of num? (answers in radians) |
| `Math.tan( num )` | What is the tangent value of num? (answers in radians) |
| `Math.asin( num )` | What is the inverse sine value of num? (assumes input of radians) |
| `Math.acos( num )` | What is the inverse cosine value of num? (assumes input of radians) |
| `Math.atan( num )` | What is the inverse tangent value of num? (assumes input of radians) |
| `Math.hypot( a, b )` | Given sides of length a and b, what is the hypotenuse? |
| `Math.toDegrees( r )` | Convert r radians to degrees |
| `Math.toRadians( d )` | Convert d degrees to radians |

Example usage...

```java
class Main {
    public static void main(String[] args) {
        double a = Math.pow(2.0, 5.0); // Find the exponent of 2 to the power of 5
        System.out.println(a);
    }
}
```

## Problems

The following questions assume you will use variables as the inputs into each problem, so the problems can recalculate solutions by changing the value assigned to the variable. You should also print the given information in your answer.

For example, if the question was, “for a given number of minutes, how many whole hours are there?”, the pseudo code for the solution could look like:

```
# Input
MINUTES = 310
# Calculation
LEFTOVER_MINUTES = MINUTES % 60
HOURS = (MINUTES - LEFTOVER_MINUTES) / 60
PRINT "In "+ MINUTES +" minutes, there are "+ HOURS +" full hours"
```

1. For any given number, extract the 10s digit. For example, The tens digit in 1234 is 3.

2. Area of a right angled triangle calculator. Given values for base and height, print the area.

3. For any two digit number, swap the position of the digits. For instance, 79 becomes 97.

4. For any three digit number, print the sum of the three digits. For instance 273 becomes 12 (2+7+3)

5. For any given year, print the century that year belongs to. Remember that 1999 and 2000 were the 20th century, whereas 2001 was the beginning of the 21st century.

6. Given a number representing the number of seconds since midnight, print the time in 24hour clock format. For example 70500 seconds should print a time of 19:35.

7. Area of a non-right angled triangle calculator. Given values for length a, length b and angle in degrees C, return the area of the triangle (remember you will have to convert degrees to radians first).

8. For any given values for a, b and c, will provide the solutions to the quadratic formula (you may assume both solutions are required). Be careful with your order of precedence. Here is an example solution set for testing: If y=2x^2-4x-10 then the solutions are 3.44949 and -1.44949.

9. Surface area of a sphere calculator given a radius

10. Volume of a cone calculator

11. Work out the Java order of operations by testing the following `3/4+5*2/33-3+8*3` .

## 4. Strings

A string is the programming term for text. More properly it can be thought of as a string of characters. We've already used strings whenever we've used the double quote enclosed text in the println() function. We can also use strings with variables. The string variable in Java is defined with a capitalised String.

The generalised rule of creating a String variable is...

```
String variableName = "initial value";
```

That said, Java does not require you to provide an initial value if you choose not to. You can declare and create the variable without that, and then set the value later. This would be valid...

```
String variableName;
variableName = "some value";
```

You only use the String keyword when creating the variable. When changing its value in later code you do not use the keyword.

The following example will create two String variables. Before coding this, predict what the printed message will be. Can you identify the one minor thing that you might want to alter before programming it?

```
class Main {
    public static void main(String[] args) {
        String name = "Mr Baumgarten";
        String message = "Hello" + name + "!";
        System.out.println(message);
    }
}
```

Second prediction: What will be the printed message in this version?

```
class Main {
    public static void main(String[] args) {
        String name, message;
        name = "Mr Baumgarten";
        message = "Hello" + name + "!";
        name = "Mr B";
        System.out.println(message);
    }
}
```

## Input strings

Like numbers, we can easily prompt the user to enter string information into our program.

```java
import java.util.Scanner;                                 // Import Scanner object

class Main {
    public static void main(String[] args) {
        String name, message;
        System.out.println("What is your name?");
        Scanner keyboard = new Scanner(System.in); // Create keyboard reading object
        name = keyboard.nextLine();                // Read a line from the keyboard
        message = "Hello, "+name+"!";
        System.out.println(message);
    }
}
```

## String functions

There are a number of different functions available for you to inspect and manipulate the content of strings. Some are more intuitive than others. Take the time to try each of these out and determine what they do and post your responses to the relevant Google Classroom assignment.

```java
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Type something:");
        String s1 = keyboard.readLine();

        // The .length() function
        System.out.println( s1.length() );

        // The .charAt() function
        System.out.println( s1.charAt(0) );
        System.out.println( s1.charAt(1) );
        System.out.println( s1.charAt(2) );

        // The .charAt() with a twist. This may take some figuring out
        System.out.println( s1.charAt(0) + 0 );

        // .indexOf() and .lastIndexOf()
        System.out.println( s1.indexOf(" ") );
        System.out.println( s1.lastIndexOf(" ") );

        // .substring() - try changing the numbers.
        System.out.println( s1.substring(8, 12) );

        // .replace()
        System.out.println( s1.replace(" ", "#") );

        // Hopefully these last two are intuitive
        System.out.println( s1.toUpperCase() );
        System.out.println( s1.toLowerCase() );
    }
}
```

Test your understanding of these functions by completing the problem set.

## Problems

1. Given a string, return a new string made of 3 copies of the last 2 chars of the original string. Assume the input string length will be at least 2 characters. For example, the string “`Hello`” should result in “`lololo`”.

2. Given a string, return the string made of its first two chars, so the String “`Hello`” yields “`He`”. If the string is shorter than length 2, return whatever there is, so “X” yields “X”, and the empty string "" yields the empty string "".

3. Given a string, return a version without the first and last char, so “`Hello`” yields “`ell`”. The string length will be at least 2.

4. Given 2 strings, return their concatenation, except omit the first char of each. The strings will be at least length 1. For example, strings “`Hello`” and “`There`” should result in “`ellohere`”.

5. How would you print the following? `All "good" people should come to the aid of their country`. (ie: you’ll have to research how to print the double quote character)

6. Write code that will produce the following printout using only a single println().

```
Hello
Hello again
```

7. Write code that will produce the following printout.

```
A backslash looks like this \, ...right?
```

8. What is output by the following?

```
String pq = "Eddie Haskel";
int hm = pq.length();
String ed = pq.substring(hm - 4);
System.out.println(ed);
```

10. Given a string input of a date in format, `dd/mm/yyyy`, print an output advising the current day, month and year number.

## Casting between data types

| FROM / TO | int | long | double | String |
|---|---|---|---|---|
| int | | =(int)num; | =(int)num; | =Integer.parseInt(str); |
| long | =(long)num; | | =(long)num; | =Long.parseLong(str); |
| double | =(double)num; | =(double)num; | | =Double.parseDouble(str); |
| String | =Integer.toString(num); | =Long.toString(num); | =Double.toString(num); | |

Because variables in Java are strictly typed, a common task is to need to convert the value in one variable to that of another type.

For instance, if we have a String that contains the text of a number, we need to convert it to a numeric variable before we can perform calculations on it. This process of conversion is known as casting.

The table shows you how to cast between the common data types.

For example… to convert a number that is contained within a string to a long, so you can perform a calculation on it, and then convert it back again…

```
class Main {
    public static void main(String[] args) {
        String s1 = "12345";
        long a = Long.parseLong(s1);
        a = a * 2;
        s1 = Long.toString(a);
        System.out.println(s1);
    }
}
```

# 5. Selection

The power of programming comes from letting the computer do work for us. To do that it needs to make decisions. We can have Java make decisions on the basis of comparing one value to another. These comparisons should always generate a boolean result, that is true or false. We can then instruct Java to execute code based on whether a comparison is true or false.

## Numeric comparisons

To compare the values of any numeric values or variables, the following operators exist:

```
if (a == b) {     // ... is equal to
if (a != b) {     // ... is not equal to
if (a > b) {      // ... greater than
if (a >= b) {     // ... greater than or equal to
if (a < b) {      // ... less than
if (a <= b) {     // ... less than or equal to
```

Note that a and b can either be a value or a variable, and that the surrounding set of parenthesis is required.

For example...

```
boolean result;
int a = 10, b = 3;
result = ( a == b );
System.out.println( result );
result = ( a != b );
System.out.println( result );
result = ( a > b );
System.out.println( result );
result = ( a < b );
System.out.println( result );
```

Take careful note of the difference in punctuation between setting a value to a variable, and comparing two values! Setting a variable to a given value uses the single equal sign = whereas comparing two values or variables uses the double equal sign ==.

## String comparisons

There are a variety of functions suitable for comparing the values of strings as the following illustrates:

```
java.util.Scanner keyb = new java.util.Scanner(System.in);
System.out.print("Type string 1: ");
String s1 = keyb.nextLine();
System.out.print("Type string 2: ");
String s2 = keyb.nextLine();

System.out.println( s1.equals(s2) ); // also Objects.equals(s1, s2)
System.out.println( s1.compareTo(s2) );
System.out.println( s1.contains(s2) );
System.out.println( s1.endsWith(s2) );
System.out.println( s1.startsWith(s2) );
System.out.println( s1.isEmpty() );
```

You will notice they all return boolean results except .compareTo() which returns an integer result indicating how closely the two strings compare or differ. Specifically, it will return:

- zero when string values match
- negative when the parent object (s1) is alphabetically before the parameter (s2)
- positive when the parent object (s1) is alphabetically after the parameter (s2)

(The size of the positive or negative number indicates the value of the number of characters different. For example "a".compareTo("d") will return -3 because the letter a is three values before the letter d)

## Multiple comparisons

Boolean logic can be used to daisy chain multiple comparisons into one instruction.

- The AND operator is the double ampersand &&.
- The OR operator is the double pipe ||.
- The NOT operator is the exclamation !.

The following is a valid example:

```
int a = 13;
int b = 4;
int c = 10;
boolean result;

result = ( (a > b) && (a < c) );
result = ( (a > b) || (a < c) );
result = ( (a != b) && (a < c) || (b == c) );
```

Where you are uncertain about order of precedence, it is recommended to use an additional set of parentheses to enforce your intended outcome.

Having determined how to have Java compare values one against the other, we can now build on that by using if statements to selectively execute code based on the result of the comparison.

The overall syntax is:

```
if ( condition ) {
   doSomething;
   doSomething;
} else if ( condition ) {
   doSomething;
   doSomething;
} else {
   doSomething;
   doSomething;
}
```

Let's make a simple example:

```
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        boolean result;
        Scanner keyb = new Scanner(System.in);
        System.out.print("a: ");
        double a = keyb.nextDouble();
        System.out.print("b: ");
        double b = keyb.nextDouble();

        if ( a == b ) {
            System.out.println("a and b are equal");
        } else if (a < b) {
            System.out.println("a is less than b");
        } else {
            System.out.println("a is greater than b");
        }
    }
}
```

## Problems

1. Create a program that asks for two people's names and their ages. Print the name of the oldest person (or if they might be twins?!)

   ```
   Person 1 name: Jack
   Person 1 age: 16
   Person 2 name: Mary
   Person 2 age: 17
   Mary is older
   ```

2. Create a program that will input three integers and print the highest of them.

3. Suppose you ask the user what the temperature is. Create a program that will respond as follows:

   If the temperature is between 20 and 27, say that it is "Just right"
   If the temperature is below 20, say that it is "too cold"
   If the temperature is above 27, say that it is "too hot"

4. Create a program that allows the user to input the sides of any triangle, and then print True/False to indicate if the triangle is a Pythagorean Triple or not.

5. Create a program that will input two Strings and alert if they have the same last character.

6. Write a program to check if a triangle is equilateral, isosceles or scalene. An equilateral triangle is a triangle in which all three sides are equal. A scalene triangle is a triangle that has three unequal sides. An isosceles triangle is a triangle with (at least) two equal sides.

7. Create a program that will input three Strings and will print them out into alphabetical order.

   ```
   String 1: hi
   String 2: there
   String 3: have a great day
   have a great day
   hi
   there
   ```

# 6. Iteration

Another term for iteration is repetition. It is where we ask Java to repetitively execute a block of code while a condition is being met. There are two main types, the while loop and the for loop.

The basic syntax of a "while loop" is as follows:

```
while ( comparison ) {
   instructions();
}
```

The following is a simple example of the while() loop that will count from 0 to 9.

```
int a = 0;
while ( a < 10 ) {
   System.out.println( a );
   a = a + 1;
}
```

The basic syntax of a "for loop" is:

```
for ( initialization ; comparison ; iterationIncrementer ) {
   instructions();
}
```

Here is the for() loop counting from 0 to 9

```
for (int i=0 ; i<10 ; i=i+1 ) {
   System.out.println( i );
}
```

## Problems

1. Write a program which will find all such numbers which are divisible by 7 but are not a multiple of 5, between 2000 and 3200 (both included).

2. Write a program which iterates the integers from 1 to 50. For multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

3. Write a program to construct the following pattern, using a nested for loop.

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

4. The fibonacci sequence is created by summing the two previous numbers together. The first 10 numbers in the sequence are 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.

   Use a loop to create a program that will calculate the n-th number of the sequence. For instance, if asked for the 8th number, it should provide the answer of 21.

5. Write a program to check the validity of password input by users. The rules for a valid password are:

   At least 1 letter between [a-z] and 1 letter between [A-Z].
   At least 1 number between [0-9].
   At least 1 character from [$#@].
   Minimum length 6 characters.
   Maximum length 16 characters.

6. Write a program that will allow a user to input his name. The prompt and input data would look something like this:

   Please enter your name: Peter Ustinov.

   Using a for-loop and the String method substring() print the reversal of the name. For example, the name Peter Ustinov would be: vonitsu retep. Ensure that the printout is in all lower-case.

7. Create a simple number guessing game. The program needs to work as follows:

   The computer picks a random number and stores it as a secret number (random numbers help)
   Ask the user to guess the number
   If the guess is higher than the secret number, print the message "too high"
   If the guess is lower than the secret number, print the message "too low"
   If the guess is correct, print the message "you are correct!"
   To use a while loop to keep the game going until the correct guess has been made
   Bonus points: Can you keep count of the number of guesses it takes the player to get it correct?

# 7. Functions

We have been using a lot of various functions that exist within Java already but haven't created any of our own, other than using `main()`.

Functions are a useful way of abstracting complexity in our project. Functions will generally require one or more inputs, and then provide a returning result.

Functions are blocks of code that you assign a name to. You can use that name to easily run that code again whenever you need.

Functions are very useful for separating common tasks out from your main code. It allows you to avoid repeating yourself all the time which makes your code easier to maintain. Tasks like reading from a file, saving to a file, etc are all ideally suited to being chopped off into a separate function.

> Think of an Icecreamary
>
> Lots of different possible flavours, toppings, numbers of scoops, choice of waffle or regular cone, etc.
>
> One person could order a double scoop of chocolate fudge and vanilla on a waffle cone, where as the next customer might ask for a cup of raspberry sorbet with nut sprinkles. The salesperson calculates the cost for each and advises each customer on the price. In order to calculate that cost there are a number of inputs (number of scoops, type of cone, etc) and an output (price). How it is actually calculated is not important, provided it is trustworthy and works reliably.
>
> In this way a function can provide a "black box" model through which we can create an abstraction to represent our problem.

Programmers need to know how to:

a. use other peoples abstractions and
b. be able to create their own.

For now, the abstraction we are concerned with is creating a function.

This is an example function that will convert the mathematical function $A = \pi r^2$ to Java:

```java
public static double getAreaOfCircle( double radius ) {
    double val = Math.PI * radius * radius
    return val;
}
```

Let's look at this bit by bit. Don't code it yet.

- `public` - is known as an access modifier. We'll discuss the role of access modifiers more when we look at classes. For now just make sure you specify it when you create a function.
- `static` - again understanding the function of this keyword will be further explained in the classes section.
- `double` - this is the data type that the function will return. Functions can provide a value back to the code that calls it, and the function must specify what type of data it will return.
- `getAreaOfCircle` - this is the name we are assigning the function.
- `(double radius)` - immediately following the name of the function is the list of parameter values we will expect to be supplied to the function. In this case, we are expecting one value of type double, which we will refer to internally within the function via the name radius. This name does not have any relation to any variable that may be used outside the function. To accept more than one parameter, we comma separate them.
- `{ .... }` - The code to be executed by our function is enclosed within the braces.
- `return` - This is where we specify the value to be returned to the code that called the function. Once Java encounters a return statement, it will exit the function regardless of any further code you may have written. Your function must provide a return value unless you specify the function data type as void (like we do with main).

## Your 1st function

```java
import java.util.Scanner;

class Main {
    public static double getAreaOfCircle(double radius) {
        double val = Math.PI * radius * radius;
        return val;
    }

    public static void main(String[] args) {
        Scanner keyb = new Scanner(System.in);
        System.out.println("Radius? ");
        double r = key.nextDouble();
        double a = getAreaOfCircle(r);     // Run our new function
        System.out.println(a);
    }
}
```

## Your 2nd function

This next example illustrates two ideas:

- Creating a function that requires two parameters.
- Using a function we created elsewhere. We will reuse our previous areaOfCircle function without having to rewrite it. This is one of the coolest reasons to write functions, we can start reusing previous code!

```
import java.util.Scanner;

class Main {
    public static double getAreaOfCircle(double radius) {
        double val = Math.PI * radius * radius;
        return val;
    }

    public static double getAreaOfCylinder(double radius, double length) {
        return length * areaOfCircle(radius);
    }

    public static void main(String[] args) {
        Scanner keyb = new Scanner(System.in);
        System.out.println("Radius? ");
        double r = key.nextDouble();
        System.out.println("Length? ");
        double l = key.nextDouble();
        double a = getAreaOfCylinder(r, l); // Run our new function
        System.out.println(a);
    }
}
```

## User input validation functions

Functions can be a handy way to require the user to comply with our wishes to enter information in a particular manner.

By the time we write the checking/validation code and the loop, user input checks can run to several lines, and it would be quite common within a simple program to want to validate the same style of input several times. Functions make a handy way to reuse code for this purpose.

For instance, study the following and predict its behaviour before coding it.

```
public static String confirm( String prompt ) {
    String response = ""
    java.util.Scanner keyb = new java.util.Scanner(System.in);
    while (true) {                  // what will `while (true)` do?
        System.out.println( prompt );
        response = keyb.nextLine();
        if (response.equals("y") || response.equals("n")) {
            return response;        // note the `return` is not at the end!
        } else {
            System.out.println("Only 'y' or 'n' are accepted. Please try again.");
        }
    }
}
```

## Problems

1. Create a function `getAreaRightAngleTriangle(base, height)` that returns the calculated area.

2. Create a function `getAreaTriangle(base, height, angle)` that returns the calculated area for a non-right angled triangle (remember you will need to convert the angle to radians before using it with the sine function).

3. Create a user input validation function that requires the input of a date in the `dd/mm/yyyy` format. Bonus points if you ensure that the dd, mm and yyyy values make sense (ie: day should be between 1 and 31).

# 8. Arrays

Note: In this section I will limit discussion to using arrays of primitive data types of one dimension. Two dimensional arrays, and arrays of custom objects will be addressed later.

So far we've talked about numbers, strings and Booleans where each variable just stores one thing at a time.

What happens if we want to manage a shopping list? or a list of students in my class... and we want to be able to manage that entire list of things together? Java allows us to do this by creating arrays. So instead of creating variables student1, student2, student3, etc, we can have one variable called students and use that in our code.

There are two main types of arrays in Java. The classic static array has a size that is fixed at declaration. Once the memory space has been allocated, they can not be resized. The other type is known as ArrayLists and is a dynamically resizable construct available when the size of the array is not known in advance. For now, we will focus on the static array.

## Why use arrays?

It might be easiest to think of an array as the technical term for a list. It allows us to store a set of values all assigned to one variable identifier. They are very useful when we have a collection of values that are similar in nature and that will be processed in the same manner.

For example, what if we are supposed to keep records of test scores obtained by a group of students. Without a list we could use something like the following:

```java
double score1 = 59.0;
double score2 = 92.0;
double score3 = 85.0;
double score4 = 61.0;
double score5 = 78.0;
```

To then calculate the highest, lowest and average score? That would look like...

```java
double highest = score1;        // Initially set highest to the first value
if (score2 > highest) { highest = score2; }
if (score3 > highest) { highest = score3; }
if (score4 > highest) { highest = score4; }
if (score5 > highest) { highest = score5; }
double lowest = score1;        // Initially set lowest to the first value
if (score2 < lowest) { lowest = score2; }
if (score3 < lowest) { lowest = score3; }
if (score4 < lowest) { lowest = score4; }
if (score5 < lowest) { lowest = score5; }
double average = (score1 + score2 + score3 + score4 + score5) / 5
System.out.println("The highest score was "+highest+", the lowest was "+lowest+" and
the average was "+average);
```

You can see that the whole process will quickly get very tedious. There will be a lot of copy-and-pasting-and-renaming of code going on. Imagine if we needed to scale this up to 100 students for an entire year group? Unmanageable and error prone!

Enter the array!

The equivalent task using an array might look like

```
double[] scores = {59.0, 92.0, 85.0, 61.0, 78.0};
double highest = scores[0];     // Initially set highest to the first value
double lowest = scores[0];      // Initially set lowest to the first value
double total = 0;               // Running total for calculating the average later
for (double val : scores) {     // Iterate through each `value` within `scores`
    if (val > highest) {
        highest = val;
    }
    if (val < lowest) {
        lowest = val;
    }
    total = total + value;
}
double average = total / scores.length;
System.out.println("The highest score was "+highest+", the lowest was "+lowest+" and 
the average was "+average);
```

Our scores array can easily contain 1000s of records and we would not have to change a single line of the calculations code! Arrays can be extremely useful!

## Declaring an array

We'll start by looking at the static array. There are two methods to declare a static array.

Method 1

```
int[] primes = new int[10];
primes[0] = 1;
primes[1] = 2;
primes[2] = 3;
primes[3] = 5;
primes[4] = 7;
primes[5] = 11;
primes[6] = 13;
primes[7] = 17;
primes[8] = 19;
primes[9] = 23;
```

Method 2

```
int[] primes = {1,2,3,5,7,11,13,17,19,23};
```

## Iterating arrays

There is a special "for loop" for iterating through an array. The following two loops produce the same output.

The for loop you are used to...

```
for (int i=0; i<primes.length; i++) {
   System.out.println( primes[i] );
}
```

The special for loop for arrays...

```
for (int item : primes) {
   System.out.println( item );
}
```

## Functions and properties of arrays

- These functions require you to `import java.util.Arrays;`

Check if two arrays are filled with matching values

```
if ( Arrays.equals( primes, other )) {
    System.out.println("The two arrays match");
}
```

Length of an array

```
int len = primes.length;
```

Sort an array in ascending order

```
Arrays.sort( primes );
```

Create a string listing the contents of the array

```
// will output: [1, 2, 3, 5, 7, 11, 13, 17, 19, 23]
System.out.println( Arrays.toString( primes ));
```

## Problems

For these problems, I used https://www.random.org/integer-sets to create 2 sets of unsorted random integers, and https://www.randomlists.com/random-words to create the list of words. You can use my sets below, or create your own.

```
int[] numbers1 = {936, 489, 845, 959, 550, 687, 776, 604, 244, 694, 546, 322, 753, 1000,
294, 18, 405, 271, 555, 759};
int[] numbers2 = {989, 463, 472, 730, 1, 419, 591, 185, 884, 318, 547, 222, 605, 71, 468,
451, 310, 407, 498, 132};
String[] words1 = {"wave", "room", "gentle", "search", "true", "board", "fowl", "upbeat",
"name", "hug", "vengeful", "observe"};
```

Exercises:

1. Given an array of integers, sum all integers and find the average value.
2. Given an array of integers, find the maximum and minimum number in the array.
3. Given an array of strings, print all items with length of at least 5 characters, and where the first and last character do not match.
4. Given an array of integers, print any duplicates in the array (ie: values that appear more than once).
5. Given two arrays of integers, print any values that exist in both arrays.
6. Given two arrays of integers, print any values that only appear in one array.
7. Given an array of integers, reverse the array and print the result. ie: The array { 6, 21, 10, 13 } should be converted to { 13, 10, 21, 6 }. ****CHALLENGE****
8. Given an array of integers, sort the array into ascending order and print the result (without using Arrays.sort()). ****CHALLENGE****

# 9. Exceptions

An exception is a critical event that should be foreseeable by a programmer that Java expects you to program a response to. Examples situations include:

- Attempting to divide by zero
- Attempting to cast a string to an integer when it doesn't contain a number
- Attempting to read a file beyond its end point
- Attempting to write to a file that is read-only, or has a full disk, or has other issues
- Attempting to access an array beyond the number of elements it contains

Let's make an example:

```java
import java.util.Scanner;

public class Main{
    public static void main( String args[] ) {
        Scanner keyb = new Scanner(System.in);
        // First input
        System.out.print("Input a number:");
        String s1 = keyb.nextLine();
        int i1 = Integer.parseInt( s1 );
        // Second input
        System.out.print("Input another number:");
        String s2 = keyb.nextLine();
        int i2 = Integer.parseInt( s2 );
        // Perform division
        int result = i1/i2;
        // Output result
        System.out.println(result);
    }
}
```

The above code will compile and execute just fine until....

- What happens if you enter "ten" as one of the inputs?
- What happens if you enter "0" as the second input?

The first will give you the message, `Exception in thread "main" java.lang.NumberFormatException`.

The second should give you the message, `Exception in thread "main" java.lang.ArithmeticException: / by zero`

Java expects us to catch these exceptions so our program can deal with them gracefully instead of causing the program to fail.

We do this by adding the try / catch construct to our program.

The general rule looks like...

```
try {
   // do something that could generate an error
} catch (Exception e) {
  // code how to respond to an error
}
```

The following illustrates how to fix it through the simple use of try and catch.

```
import java.util.Scanner;
public class Main{
    public static void main( String args[] ) {
        Scanner keyb = new Scanner(System.in);

        try {
            // Input 1st number
            System.out.print("Input a number:");
            String s1 = keyb.nextLine();
            int i1 = Integer.parseInt( s1 );
            // Input 2nd number
            System.out.print("Input another number:");
            String s2 = keyb.nextLine();
            int i2 = Integer.parseInt( s2 );
            // Perform division
            int result = i1/i2;
            // Output result
            System.out.println(result);
        } catch(Exception e) {
            System.out.println("An exception happened.");
            System.out.println("Exception type: "+e.getClass().toString());
            System.out.println("Message:        "+e.getMessage());
        }
    }
}
```

A better solution is to check for the type of exception so a more meaningful message can be given to the user.

```java
import java.util.Scanner;

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

        try {
            // Input 1st number
            System.out.print("Input a number:");
            String s1 = keyb.nextLine();
            int i1 = Integer.parseInt( s1 );
            // Input 2nd number
            System.out.print("Input another number:");
            String s2 = keyb.nextLine();
            int i2 = Integer.parseInt( s2 );
            // Perform division
            int result = i1/i2;
            // Output result
            System.out.println(result);
        } catch(NumberFormatException e) {
            System.out.println("An input could not be converted to an integer");
        } catch(ArithmeticException e) {
            System.out.println("You attempted to divide by zero");
        } catch(Exception e) {
            System.out.println("An unknown exception happened.");
            System.out.println("Exception type: "+e.getClass().toString());
            System.out.println("Message:        "+e.getMessage());
        }
    }
}
```

To use this second method, we need to know the individual exception types. This is more work but it allows for a more precise response to the error.

Not sure what the likely exceptions are called? The simple way is to not catch them, then run your code so as to generate the exception, and then Java will tell you the exception names!

Finally, a more complete solution would be to actually check for exceptions at each point they could occur and to provide the user a way to fix their inputs if possible.

The following shows this.

```java
import java.util.Scanner;

public class Main{
    public static void main( String args[] ) {
        Scanner keyb = new Scanner(System.in);
        int i1, i2;

        // Input and convert first number
        while (true) {
            try {
                System.out.print("Input a number:");
                String s1 = reader.nextLine();
                i1 = Integer.parseInt( s1 );
                break;  // break out of the while loop
            } catch (NumberFormatException e) {
                System.out.println("Not an integer. Please try again.");
            }
        }

        // Input and convert second number
        while (true) {
            try {
                System.out.print("Input another number:");
                String s2 = reader.nextLine();
                i2 = Integer.parseInt( s2 );
                break;  // break out of the while loop
            } catch (NumberFormatException e) {
                System.out.println("Not an integer. Please try again.");
            }
        }

        // Perform division
        try {
            int result = i1/i2;
            System.out.println(result);
        } catch (ArithmeticException e) {
            System.out.println("Can not divide by zero");
        }
    }
}
```

You can see here that using a function would be the better way to input the number and convert it to an integer with the try/catch. It would obey the rule of `DRY - don't repeat yourself`.

# 10. Files

Reading and writing files in Java is a bit problematic. It is far more complicated than it needs to be. A considerable part of the problem is that as soon as you start searching for anything to deal with file handling in Java, you will quickly encounter a lot of different ways of doing it. Just to prove the point, these are all valid yet different ways to read files in Java:

- `java.io.FileReader.read()` - reads in one character at a time, without any buffering. It's meant for reading text files.
- `java.io.BufferedReader.readLine()` - reads an entire line at a time,
- `java.io.FileInputStream.read()` - reads in one byte at a time, without any buffering. While it's meant for reading binary files such as images or audio files, it can still be used to read text files.
- `java.io.BufferedInputStream.read()` - reads a set of bytes all at once into an internal byte array buffer
- `java.nio.file.Files.readAllBytes()` - The Files class is part of the new Java I/O classes introduced in jdk1.7. It only has static utility methods for working with files and directories.
- `java.nio.file.Files.readAllLines()` - The Files class is part of the new Java I/O classes introduced in jdk1.7. It only has static utility methods for working with files and directories.
- `java.nio.file.Files.lines()` - The Files class is part of the new Java I/O classes introduced in jdk1.7. It only has static utility methods for working with files and directories.
- `java.util.Scanner.nextLine()` - can be used to read from files or from the console (user input)
- `org.apache.commons.io.FileUtils.readLines()` – Apache Commons
- `com.google.common.io.Files.readLines()` – Google Guava

Which method is "most" correct? Which method do you use when?

To be honest, at this point in your programming journey, don't worry too much about speed or memory efficiency issues, find a method that seems "logical" to you and stick with it.

> *If you really want to learn about any of the above methods that I won't discuss here, check this good resource, https://funnelgarden.com/java_read_file/*

The following are some simple "recipes" I suggest you use to get started.

## Reading a text file

```java
import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.util.List;

public class Main {
  public static void main(String [] args) {
      try {
          String fileName = "test.txt";
          // Open the file
          File file = new File(fileName);
          // Load the content of the file assuming UTF-8
          List<String> dataList = Files.readAllLines(file.toPath(),
             StandardCharsets.UTF_8);
          // Convert the list to an array
          String[] dataArray = dataList.toArray();
          // Use the array like normal
          for(String line : dataArray) {
              System.out.println(line);
          }
      } catch (Exception e) {
        System.out.println(e.getMessage());
      }
  }
}
```

## Writing a text file

```java
import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.*; // import it all
import java.util.*; // import it all

public class Main {
    public static void main(String [] args) {
        Scanner keyb = new Scanner(System.in);
        String[] data = new String[10];
        for (int i=0; i<10; i++) {
            System.out.print("Type line #"+i+":");
            data[i] = keyb.nextLine();
        }
        try {
            String fileName = "test2.txt";
            // Open the file
            Path path = Paths.get(fileName);
            // Convert array to list
            List<String> dataList = Arrays.asList(data);
            // Write to file
            Files.write(path,dataList,Charset.defaultCharset());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

## Folder walk

If you want a list of files in a folder, Java can help you out with that too.

```java
// From: https://stackoverflow.com/a/2056326
import java.io.File;

public class Main {

    public static void walk( String path ) {
        File root = new File( path );
        File[] list = root.listFiles();
        if (list == null) return;
        for ( File f : list ) {
            if ( f.isDirectory() ) {
                System.out.println( "Dir:" + f.getAbsoluteFile() );
                walk( f.getAbsolutePath() );
            }
            else {
                System.out.println( "File:" + f.getAbsoluteFile() );
            }
        }
    }

    public static void main(String[] args) {
        walk("c:\\" );
    }
}
```

## Problems

1. Read a text file into a list of strings, and print out the number of lines in the file.
2. Ask the user for the name of a file they'd like to create. Using a while loop, keep ask the user to type an input and only stop when they enter an empty input. Save all the lines entered as the content of the file.
3. Read a list of strings from a text file. Tell the user how many lines there are and ask them to enter a line number indicating one they would like to read. Print just the content of that line to the user.
4. Read a list of strings from a text file. Tell the user how many lines there are and ask them to enter a line number indicating one they would like to change. Prompt the user for the new content of the relevant line. Write to the file the new list of strings.