Static methods.

Software is never built as a monolithic piece of code – it would make it extremely inflexible.

Methods – functional building blocks. Used to implement a functionally independent task. Consider methods of the String class: substring, indexOf, length.

Building an application using simple well designed and documented methods
- makes it easier to write, read, test, and modify software
- allows for distribution of programming tasks between people
- creates reusable components

We’ll start by looking at static methods of Java.

All Java methods are defined within a class.

All Java methods have
- name
- zero or more input parameters and
- one or no return value (output).

Methods may also have
- Local variables and constants
Example:

```java
/*
 * Static methods demo.
 * A class that contains simple circle calculation methods.
 *
 */

public class CircleCalc {

    public static final double PI = 3.1415926;

    public static void main (String args []) {
        double radius;
        System.out.println("This program makes simple circle calculations");
        System.out.println("Please enter the radius if a circle");
        radius = SavitchIn.readLineDouble();

        double cArea;
        // invoke the calcArea method defined below
        cArea = calcArea (radius);
        // same as cArea = CircleCalc.calcArea(radius)

        System.out.println("The area of that circle is" + cArea);

        // here, invoke the method within an expression.
        System.out.println("The circumference is " + circumference(radius));
    }

    /*
    * double calcArea(double radius) - returns area of circle
    */
    public static double calcArea(double radius){
        double area;
        area = PI*radius*radius;

        return (area);
    }

    /*
    * double circumference (double r) - returns circumference of circle with radius r
    */
    public static double circumference (double r){
        return (2*PI*r);
    }

    }
}
Method declaration and invocation

- A *method declaration* specifies the code that will be executed when the method is invoked (or called)

- A method declaration begins with a method header:

  ```java
  public static double calcArea (double radius)
  ```

  - **return type**, i.e. type of return value  
  - **name** type and name of formal parameter (aka argument)

- Followed by method body in `{ … }`

  ```java
  double area;  // area is a local variable (declared inside method body)
  area = PI*radius*radius;
  return (area);  // value returned from the method
  ```

- When a method is invoked, the flow of control jumps to the method and executes its code
- When complete, the flow returns to the place where the method was called and continues
- The invocation may or may not return a value, depending on how the method is defined. A method that does not return a value has a `void` return type.
- There are two different ways of calling a static method
  1. By name, e.g.
     ```java
     cArea = calcArea (radius);
     ```
     works only from within the same class.
  2. Using the classname as in
     ```java
     cArea = CircleCalc.calcArea (radius);
     ```
     works from within another class as well (e.g. SavitchIn.readLine() – SavitchIn is the name of the class that contains a static methods readLine() )
Parameter passing:

Let’s consider a slightly more complex example in which the method has three parameters of different types:

```java
public static char demo (int num1, int num2, String message) {
    int sum;
    char result;
    sum = num1 + num2;
    result = message.charAt (sum);
    return result;
}
```

Suppose this method is invoked in the following call where `myNum = 5`:

```java
aChar = demo (myNum, 3, “Hello World “)
```

What happens:

a. Parameters in the call are evaluated and assigned to the method’s arguments, i.e.

```
5  3  “Hello World”
  ↓  ↓  ↓
demo (int num1, int num2, String message)
```

b. The control is passed to the first statement of the method.

c. Statements are executed in order until a return statement is encountered:

```
return result;
```

Return statement terminates the execution of the method and passes control back to the point from where the method was invoked;

The returned value (i.e. value of `result`) is substituted in place of the call, i.e.

```
aChar = demo (myNum, 3, “Hello World “)
```
Notes on the return statement:

- the type of value returned must match the method’s return type, specified in the method header.
- a method may have any number of return statements, but it is a rule of good programming style to have a unique exit point from a method.
- return statement terminates the method execution.
- a method that does not return a value has a `void` return type

Scoping rules

- Local variables can be declared inside a method.
- Local variables and formal parameters are accessible only from within the method definition and are not accessible from outside of the method.
- The formal parameters of a method create *automatic local variables* when the method is invoked.
- When the method finishes, all local variables are destroyed (including the formal parameters)