PART 1

Whether you knew it or not, all the programming that you have performed until now was in the boundaries of *procedural programming*.

A procedural program is written as a list of instructions, telling the computer, step-by-step, what to do: Define a variable, read a number, multiply by 4, display something.

A typical program that you have written probably looks like:

```
class Application
{
    static double pi = 3.14;
    static void main(String args[])
    {
1      int radius = 3;
2      String color = "red";
3      double area = calculate_area(radius);
5      System.out.println("The " + color + " circle with radius " + radius + "has area " + area + ".");
    }

    static double calculate_area(int r)
    {
4       return (double) r * r * pi;
    }
}
```

The application starts by the first statement of the main method, and each line is executed sequentially. If there is a method call, then the code of the method gets executed. Later, the call again returns to the main application code.

```
class Application
{
    static double pi = 3.14;  ──────────►   Global Variable
    static void main(String args[])
    {
       int radius = 3;
       String color = “red”;
       double area = calculateArea(radius);
       System.out.println(“The “ + color + “ circle with radius “ + radius + “has area “ + area + “.”);
    }

    static double calculateArea(int r)
    {
        return (double) r * r * pi;
    }
}
```

Notice the use of the keyword static. It is used to define the variables which are global to the application. Furthermore, it is there in the declaration of each function.

But, what does static mean?

This will be best understood after you learn what an object is.

Now we will learn the very basics of object-oriented programming and objects.

As you know, the program you write in Java is inside a class.

```
class Classname
{

            // Body of the class
            // Your code goes here.

}
```

The classes you have been defining so far (the procedural programs) can be named as static classes since each method and global variable is defined to be static.

There is also another type of class which we will call a non-static class.

$$\begin{tikzcd}
 & \text{Java Classes} \arrow[dl] \arrow[dr] & \\
\text{static} & & \text{non-static}
\end{tikzcd}$$

(this you already know) (this is what you will learn)

Non-static classes will allow you to define ***objects***.

So the question is:

What is an object and why is it useful?

So far you have seen data types such as int, double, String and boolean.

Although these are essential, they are not convenient to store complicated data.

What do we mean by this ???

Suppose in your program you wanted to store the title, author and the year published of a number of books.

How would you do this?

Here are some alternatives:

Solution 1: Create one String variable for each book that holds the title, author and the year.

(We will call title, author and year the *fields* of the book.)

Example:

```
String book1 = “Lord of the Rings, J.R.R. Tolkein, 1955”;
```

Problem: Accessing each field is not easy. For example, changing the year would be cumbersome.

Solution 2: For each book, create a String variable for the title, another String variable for the author and an int variable for the year.

Example:

```
String book1Title = “Lord of The Rings”;
String book1Author = “J.R.R. Tolkein”;
int book1Year = 1955;
```

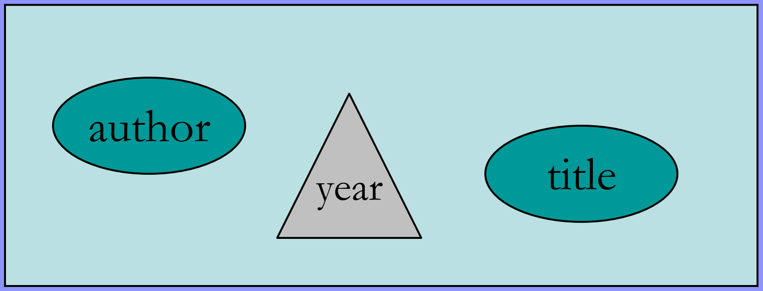
Problem: For each book, you need to create three variables! What if there were 10 fields and you wanted to store 10 books? You would have to define 100 variables.

<u>Wanted Solution:</u> "Create" a data type called Book which encapsulates three variables, String title, String author, int year, such that the following hold.

1) Accessing each of these variables is easy.
2) We only create one Book variable (object) for each book we want to store, regardless of the number of fields.

Think of the Book class as a box that includes three elements.

Book Class



Java allows you to do this.

How?

First we have to create the Book data type. This is where we use the non-static class.

Let’s create a Book data type now.

```
class Book
{
		String title;
		String author;
		int year;
}
```

This is a non-static class. The definitions of the variables do not contain the word static.

By creating this class, we have defined a new data type called Book which encapsulates three variables: title, author, year.

Now let's see how we can use this Book data type in our static class containing the main method.

Defining a Book variable:

```
class Application
{
        public static void main(String args[])
        {
                Book b1;
        }
}
```

The variable is called b1 and it is of type Book

Observe that this is no different from defining an int or a String.

```
int s;
String x
```

Initializing a variable:

```
Book b1;
b1 = new Book();
```

Or simply,

```
Book b1 = new Book();
```

Here "new Book();" creates a new Book object which has initial default values for title, author and year.

These values are title = "", author = "", year = 0.

The equality sign represents an assignment. Now the b1 variable points to a new book object.

Accessing the fields of the book and modifying their values:

We have to use the dot operator.

```
b1.title = “Lord of the Rings”;
b1.author = “J.R.R. Tolkein”;
b1.year = 1955;
```

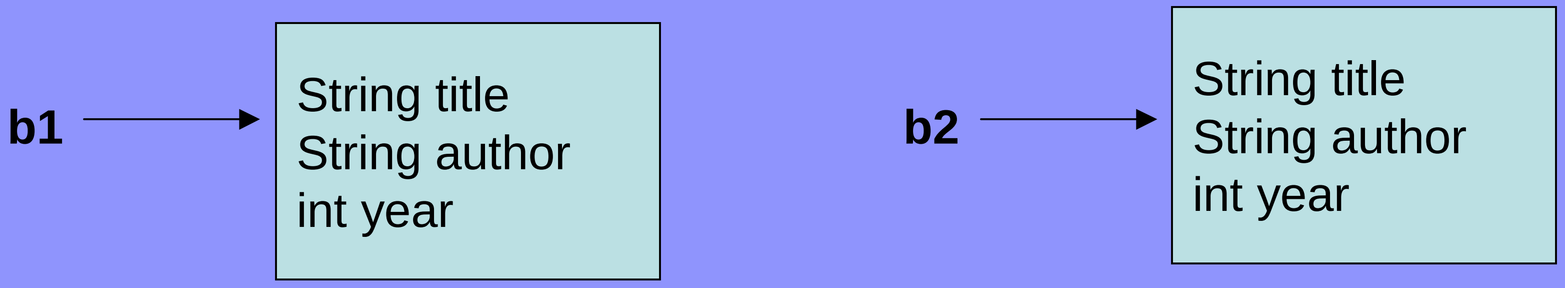
b1.title and b1.author are String variables. b1.year is an int variable. You can play with these variables just like any other String or int variable.

```
b1.title += “, The Return of the King”;
b1.year++;
System.out.println(b1.title);
System.out.println(b1.year);
```

Creating another Book object:

```
Book b2 = new Book();
b2.title = “In Cold Blood”;
b2.author = “Truman Capote”;
b2.year = 1965;
```

Each Book object we create has its own variables title, author and year. So b1.title has nothing to do with b2.title. Thus manipulating b1.title would not change b2.title and vice versa.



Returning an object:

The return type of a method can be an object and thus you can return an object in that method.

```
class Application
{
		static void main(String[] args)
		{
			Book b1 = createBook("Lord of the Rings", "J.R.R. Tolkein", 1955);
			System.out.println(b1.title);
			System.out.println(b1.author);
			System.out.println(b1.year);
		}

		static Book createBook(String t, String a, int y)
		{
			Book b = new Book();
			b.title = t;
			b.author = a;
			b.year = y;
			return b;
		}
}
```

Passing an object as a parameter:

In an application, you may have objects as parameters.

```
class Application
{
            public static void main(String[] args)
            {
                        Book b1 = new Book();
                        b1.title = “Lord of the Rings”;
                        b1.author = “J.R.R. Tolkein”;
                        b1.year = 1955;
                        print_info(b1);
            }

            static void print_info(Book b)
            {
                        System.out.println(“Title: ” + b.title);
                        System.out.println(“Author: ” +b.author);
                        System.out.println(“Year: ” + b.year);
            }
}
```

Review of Concepts

- Two types of classes: static and non-static. Non-static classes enable one to create objects.
- Objects allow you to encapsulate more than one variable in one object variable.
- The fields of an object are accessed using the dot operator.
- Each object you create has its own fields independent from the fields of other objects that may be defined.
- Objects can be returned in a method or can be passed to methods through parameters (just like int or String variables).

Review of Syntax

- Non-static class (defining a new data type)

```
class Book
{
		String title;
		String author;
		int year;
}
```

- Creating an object

```
Book b1 = new Book();
```

- Accessing the fields of an object using the dot operator

```
b1.title = “Lord of the Rings”;
b1.author = “J.R.R. Tolkein”;
b1.year = 1955;
```

Exercise

- Write a non-static class called Student that has three fields: String firstName, String lastName and int id.
- Complete the createNewStudent() method below. It is supposed to create a new student object with the given values of first name, last name and id number. Then this student should be returned.

```
class Application
{
   static void main(String[] args)
   {
      Student std = createNewStudent(“James”, “Newsted”, 110231621);
      System.out.println(std.firstName + “ ” + std.lastName + “ , ” + id);
   }
   static Student createNewStudent(String fname, String lname, int i)
   {
      // your code goes here
   }
}
```

Solution

```
class Student
{
        String firstName;
        String lastName;
        int id;
}
```

```
static Student createNewStudent(String fname, String lname, int i)
{
        Student s = new Student();
        s.firstName = fname;
        s.lastName = lname;
        s.id = i;
        return s;
}
```