**Objects, Classes, Methods, Fields (aka InstanceVariables)**

An **object** is an instance of a **class**. You "Instantiate" a class to create an object. Instantiation is usually performed using the **new** operator and a **constructor** for the class.

A **class** is a collection of: **data** (in Java these are: **fields or instance variables**) and **operations** that may be performed on the data (in Java these are **methods**). Formally, a class is the definition of a new **abstract data type** (ADT). A class is like the idea of a blueprint. With a blueprint, you can construct a house. With a class you can construct an object. Remember: **Blueprint** is to **house** as **class** is to **object**.

A **field** or **instance variable** is a variable declared outside of any method in a class. This is considered a **class-level** member. These are **global variables** that can be accessed by any of the methods in the class. The fields/instance variables are always initialized when an object of the class is created.

Compare these with **local variables**. Local variables are variables that are declared within a method. These are not accessible outside of the method and are not automatically initialized. Fields/instance variables exist for as long as an object exists. **Local variables** are created when a method is called and are destroyed when the method ends.

Both fields/instance variables and local variables may be explicitly initialized in their declaration.

The **state** of an object is the value of its field/instance variables at an instant in time. Think of it as a “snapshot” of the object’s data.

A **method** is a collection of statements that performs some task. A method implements an **algorithm**. Methods are also class-level members. Examples of things a method may do:

- Assign a value to a field/instance variable in an object. This is done using a “mutator” method (aka “set” method).
- Return the value of a field/instance variable in an object. This is done using an “accessor” method (aka “get” method).
- Perform a calculation and assign the result to a field/instance variable or return a value.
- Return a formatted string with information about the “state” of the object. A very common example of this is a “toString” method.

**Data Types**

A variable can be declared to be one of two fundamental data types:

- **value type** – these are the 8 primitive data types in Java:
  - boolean, char, byte, short, int, long, float, double
- **reference type** – these are variables that are created to store a reference (address of an object)

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Reference Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>int (x = 10;)</td>
<td>GradeBook (\text{myGradeBook} = \text{new GradeBook();})</td>
</tr>
<tr>
<td>(\text{value})</td>
<td>(\text{reference})</td>
</tr>
<tr>
<td>(10)</td>
<td>(\text{course: &quot;CIS 101&quot;})</td>
</tr>
<tr>
<td>(x)</td>
<td>(\text{GradeBook})</td>
</tr>
<tr>
<td>(\text{Equivalent to:})</td>
<td>(\text{Equivalent to:})</td>
</tr>
<tr>
<td>(\text{int} \ x;) (\text{x = 10;})</td>
<td>(\text{GradeBook} \ \text{myGradeBook};) (\text{myGradebook} = \text{new GradeBook();})</td>
</tr>
</tbody>
</table>

Remember, objects do not have names in Java.
From the illustration on the previous page, you see that when you assign a value to a value-type variable, the value is actually stored in the variable. When you are working with an object, you can't store the object directly in a variable. Instead, the object is stored somewhere else in memory and a reference to the object (its address) is stored in a reference-type variable.

Declaring and initializing a **value type** variable:

```java
double number = 123.45; // a value type is any of Java’s primitive data types
```

1. variable `number` is created. Since type is double, 8 bytes are set aside for the variable in memory
2. location of the variable is associated with the variable name `number`
3. The variable is assigned the initial value of 123.45. This is stored in the variable.

Declaring and initializing a **reference type** variable:

```java
Account myAccount = new Account(500.00); // create and set balance to $500.00.
```

1. variable `myAccount` is created. Since this is a reference variable, the size is not important. A reference variable, regardless of the type, stores a reference (aka an address). Addresses are always the same size.
2. An Account object is created.
   a. new operator creates an object by calling the constructor for the class.
   b. for this example, the constructor takes an argument of type double
   c. a constructor is a special kind of method that does not return a value. A constructor is only executed when an object is being created. It's purpose is to initialize one or more fields/instance variables in an object.
   d. the final result of the object being created is that the new operator returns a reference (the address of the Account object in memory).
3. the variable `myAccount` is assigned the reference to the Account object returned by the new operator

Referring to the variables declared above:

To get the value of `number`, you go to the variable in memory and see what value is stored there.

To get the starting balance of the Account object, you would access the object using the reference variable `myAccount`. Using the reference variable, you would access a public method in the Account class to get the balance stored in the object.

To do this, you use this syntax: `<reference-variable> . <public-method>`. For example you might use: `myAccount.getBalance();`

`myAccount` is the reference variable, `getBalance` is a public method designed to return the value of a field/instance variable in the object.

**Sample GradeBook UML class diagram:** + means public access, - means private access

<table>
<thead>
<tr>
<th>class name</th>
<th>GradeBook</th>
</tr>
</thead>
<tbody>
<tr>
<td>fields</td>
<td>- String  courseName</td>
</tr>
</tbody>
</table>
| constructors and methods | + <<constructor>> GradeBook(String course)  
+ void setCourseName(String course)  
+ String getCourseName()  
+ String displayMessage() |