Classes, Objects, Methods, Fields (aka Instance Variables)

- 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: class or instance variables) and operations that may be performed on the data (in Java these are class or instance methods). Formally, a class is the definition of a new abstract data type (ADT). A class similar to 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)
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 a reference variable, you can’t directly store an object in the variable. Instead, the object is stored somewhere else in memory and a reference to the object (it’s address) is stored in the reference 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. the location of the variable in memory is associated with the variable name number
3. The variable is initialized to the value 123.45
```

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

```java
Account myAccount = new Account(500.00); // create Account object and set it’s 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 it’s type, stores a reference (aka an address). Addresses are always the same size.
2. An Account object is created.
   a. the new operator creates an Account object by calling a constructor for the Account 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 created. it’s purpose is to initialize one or more field/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 number and myAccount declared above:
To get the value of the variable number, Java would access the variable in memory and retrieve the value stored there.

To access methods in the Account object, Java would access the reference variable myAccount. Using the reference stored there it would locate the object in memory. To access the public methods in the object you use the expression:

```java
<reference-variable> • <public-method> // the dot () is the member access operator
```

For example you might use: `System.out.println( myAccount.getBalance() );` or `myAccount.setBalance(1000.00);`

**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>
<tr>
<td>constructors and methods</td>
<td>&lt;&lt;GradeBook(String course)&gt;&gt;</td>
</tr>
<tr>
<td></td>
<td>+ void setCourseName(String course)</td>
</tr>
<tr>
<td></td>
<td>+ String getCourseName()</td>
</tr>
<tr>
<td></td>
<td>+ String displayMessage()</td>
</tr>
</tbody>
</table>