Chapter 4: Instantiable Classes

Instantiable Class vs. Program Class

- Program class has `main()` method (entry point of the program)
- Generally don't make instances of the Program class
- Instantiable class written separately from the Program class (often by a different programmer)
- Instantiable class can be used by many different Programs

Overall Format of an Instantiable Class

1. File Header comments
2. Class Header comments
3. `import` statements (if necessary — often they are not)
4. Class declaration: `public class MyClass`

5. Class body:
    - Data member declarations
    - Constructor(s)
    - Other methods

Data Members

- Declaration format: `<modifiers> <type> <name>`
- Modifiers
    - `public` vs. `private`: `public` means the data member can be directly accessed objects from another class. `private` means *only* objects from this class can directly access the data member. Data member are almost always `private` to protect the integrity of the object. `public` data members are usually used with the `static` and `final` modifiers to create “class constants”.
    - `static`: makes the data member belong to the entire class,

  rather than having a separate copy for each instance of the class.
- `final`: makes the data member constant, so that its value can never be changed.

- Type
  - can be a primitive type (e.g. `int` or `boolean`), or the name of a class (e.g. `String`).
- Name
  - the name, or "identifier", that you will use throughout the class to access the data member.
- Note that instance (non-`static`) data members are *not* given an initial value when they are declared. **All instance data members are initialized in the constructors.**
- Class (`static`) data members can be given an initial value at the time they are declared.

Constructors

- Declared with the modifier `public` and the name of the class:

  ```
  public MyClass(<parameter list>)
  ```

- `<parameter list>` used the same as in regular methods (see below)
- Main purpose is to initialize all instance data members. Some are given values from the parameter list, others are given default starting values. It is up to the designer to decide which data members are initialized in which way.
- Can be *overloaded*, just like regular methods (see below)
- Invoked using the `new` keyword:

  ```
  MyClass myObject = new MyClass(<argument list>);
  ```

  Of course, `<argument list>` must match the `<parameter list>` in type and number of elements.

Methods

- Declaration format:

  ```
  <modifiers> <return type> <name> (<parameter list>)
  ```

- Modifiers
  - `public`, `private`, and `static` have the same meanings as for data members.
  - most methods are `public`, since this is how objects from other classes interact with objects from this one.
  - `private` methods are mostly used for convenience to allow us to re-use chunks of code that we would otherwise have to copy into several methods.
  - class (`static`) methods are not associated with any given instance, and so can not access instance data members.
- Return Type
  - methods may return one piece of information:

`<return type>` specifies the data type of that piece of information.

  - may be a primitive type (e.g. `int` or `boolean`), or the name of a class (e.g. `String`).
  - if the method returns no information (which is often the case), `<return type>` = `void`.
  - data is returned from a method to the object that sent the original message.
  - data is returned by using the keyword `return` followed by the data: `return x;`

- Name
  - just the name, or "identifier", to be used to invoke this method.
- `<parameter list>`
  - a comma-separated list of variable declarations. For example:

`int x, String name, boolean b`

- values from the message arguments are copied into the corresponding parameter.
- **Note that the type of the variable is part of the paramter and NOT part of the argument!**
  parameters: `int x, String name, boolean b`
  message: `obj.myMethod(42, ‘‘Bob’’, true)`

- Methods may be *overloaded*. This means that we can re-use the same method name for two different methods, provided they have a different list of parameters. The difference can be in the order, number, or types of paramters (names are irrelevent). The methods may also have different return types, so long as they still have different paramters. The method `abs` of the `Math` class is overloaded so that it can take (and return) `int`s, `long`s, `float`s or `double`s.