Basics of Object-Orientation and the IDE Eclipse

Objects and Classes

Object. An object (in object-oriented programming) is a structure that contains:

variables (called *fields*), each of which contains a value, and

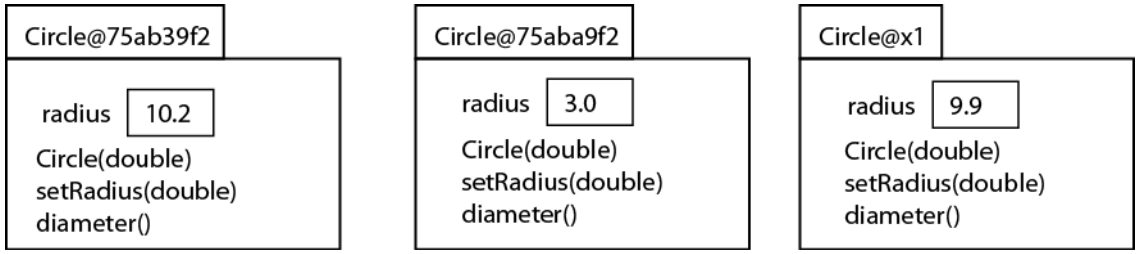
methods (*functions*, *procedures*, and *constructors*), which may use the fields to do their job. When called, functions return values, procedures perform actions, and constructors initialize the fields of a new object.

Below we show three different objects, each of which describes a circle. They contain a field radius, but with different values. They contain the same methods: (1) function diameter gives the diameter of the circle, procedure setRadius and constructor Circle both change the radius to their parameter. The complete method appears in the object, but we show only its *signature*, i.e. the name of the method and the types of its parameters.

In Java, each variable must be declared with its type before it can be used, and only values of that type can be stored in it. Java is *strongly typed*, as opposed to Matlab and Python, which are *weakly typed*.

You can imagine other functions in each object: to give the circumference and area of the circle, for example.

The names on the "tabs" of the leftmost two objects consist of the name of the "class" (Circle, see below) to which it belongs, an @ sign, and 8 hexadecimal digits, which give its address in the memory of the computer. At runtime, Java creates this name when it creates the object, and you can actually print it out. But you can't do much with it, e.g. you can't change it. The tab on the third object contains a name that *we* might give it when simply discussing objects in lecture or in section or in any discussion.



Class. A class declaration, a Java construct, is a "template" for objects of that class. Thus, the class contains *declarations* of the variables and methods that go in each object of the class. Below, we show the declaration of class Circle, which must be stored in a file named Circle.java. After the declaration, we talk about its pieces.

```
/** An object represents a circle with a radius */
public class Circle {
  private double radius; // The radius of the circle. r ≥ 0.

   /** Constructor: A circle with radius r.
       Precondition: r >= 0 */
   public Circle(double r) {
     radius= r;
   }

   /** Set the radius of the circle to r.
       Precondition: r >= 0. */
   public void setRadius(double r) {
     radius= r;
   }

   /** return the diameter of the circle. */
   public double diameter() {
     return 2*radius;
   }
}
```

We discuss this class declaration:

1. The first line "/** … **/**" is a possibly multi-line comment. Generally, each class must be preceded by such a comment that describes its object briefly. Actually, only **"/*"** is needed at the beginning; we explain the use of the second "*" much later, not in this document.

2. A class declaration has the form **public class** <class-name> { … }, where the variables and methods to appear in each object are declared within the braces { and }. Java convention: Class names start with capital letters and generally are noun phrases.

3. Keyword **public** indicates that the entity being defined with that "access modifier" can be referred to by name outside the class; keyword **private** indicates that it cannot.

4. The line starting with **private double** radius; is a declaration of a variable named radius. It can contain only **double** (floating point) values. This declaration means that every object contains this variable. The characters "//" start a comment that ends at the end of the line. Every declaration of a field must be annotated with a comment that describes the meaning of the variable and gives constraints on it. The totality of such comments on fields is called the *class invariant*.

   Some of the types we use are **int** (an integer in the range $-2^{-31}..2^{31}-1$), **double** (floating point value with a mantissa and an exponent), and **boolean** (values **true** and **false**). There are also so-called "class types", whose values are the names that are put on tabs of objects. For example, a variable declared using Circle c; can contain the value Circle@75ab39f2, which is essentially a pointer to the object in memory. All objects are referenced and used indirectly like this.

5. Following the variable declaration are three method declarations: of (1) constructor Circle, (2) procedure setRadius, and function diameter. These three declarations mean that these three methods are contained in every object of class Circle.

6. Each method declaration is preceded by a multi-line comment that describes what the method does and gives a "precondition" –constraints that the argument of a call on the method must satisfy. Every method must be preceded by such a comment; follow the style given on these three.

7. A *constructor* is recognized by the fact that it starts with **public** <class-name> (<parameter declarations>). As mentioned earlier, a constructor is used when a new object is created to initialize the fields; we don't discuss this further here.

8. A *procedure*, which does something but returns no value, has the form **public void** <procedure-name > (<parameter declarations>).

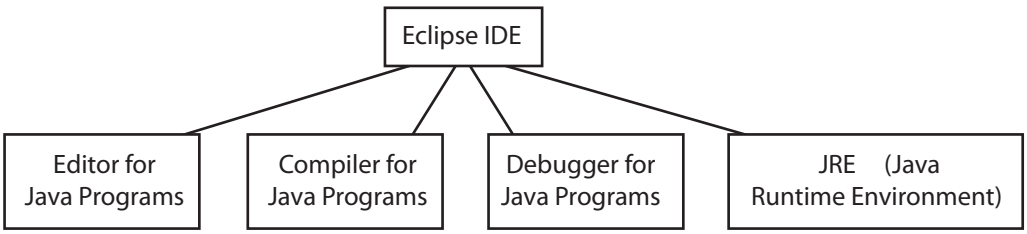
9. A *function*, which returns a value, has the form **public** <return-type> <function-name > (<parameter declarations>). The <return-type> is the type of value that the function returns.

Please note that we have not explained how to create objects of a class and save "pointers" (i.e. the names on the tabs) to them. We also haven't said how to reference variables in an object or call methods in an object. We have given you just enough so that we can talk about the application we use to write Java programs: Eclipse.

The IDE Eclipse and Java

In CS2110, we write programs using an application called *Eclipse*. We give the basics of Eclipse here. Eclipse is not just one thing but a collection of separate applications (or apps) that Eclipse manages and presents to you for your use. For example, the diagram below shows that it uses at least four other apps. You will use the editor a lot; Eclipse invokes the compiler often to check the syntax of your Java program and to translate it into the "Java Virtual Machine Language"; and it invokes the JRE when you ask it to run your program. Many other apps are involved in Eclipse. In fact, if you have an idea for one, you can write and install your own "plug-in" for it. Eclipse, then manages all these other apps and presents them to you in hopefully an effective manner. This does mean that Eclipse appears to have too many "bells and whistles" for the average programmer, but it's great for the professional.

Basics of Object-Orientation and the IDE Eclipse



Eclipse can't run a Java program unless there is a ***JRE*** (Java Runtime Environment) on your hard drive and unless Eclipse knows where it is. It also can't compile a Java program unless it knows where the compiler is on your hard drive. The compiler generally comes as part of a ***JDK*** (Java Development Kit). This is why our instructions for installing Eclipse tell you first to make sure the *JRE* and *JDK* are installed on your computer.

Downloading and Installing Java + Eclipse

Install Java before Eclipse.

Use the Java 6 Standard Edition (Java SE 6) platform or the newest one, Java SE 7. You may not use a version of Java prior to 6, such as J2SE 1.4. We will be using many new features that were introduced in Java 6, such as generics, autoboxing, and type-safe enums.

We use the latest version of Eclipse, called Juno —each major release of Eclipse is named after another moon of Jupiter.

The following webpage on the CS2110 website tells you (1) how to see if your computer has a suitable JRE and JDK, (2) how to download and install the JRE and JDK, and (3) how to download and install Eclipse.

http://www.cs.cornell.edu/courses/CS2110/2013sp/resources.html

Eclipse Terminology. Creating a Project

We now introduce the necessary terminology, as painlessly as we can.

1. **IDE**. Eclipse is an *IDE*: an Integrated Development Environment —integrated because it integrates several tools such as an editor for Java programs, a compiler, and a debugger.
2. **Workbench**. Eclipse uses this word for the whole desktop development environment. When you first start Eclipse, you should see a window with "Welcome Eclipse" on it, giving you an Overview, Samples, Tutorials, and What's New. In the top right is the word Workbench; click on it to see the Workbench.
3. **Welcome page**. To get back to the Welcome page, use menu item Help → Welcome. The Tutorial on creating a Hello World Application can be helpful.
4. **Perspective**. A *perspective* defines set and layout of views in the Workbench. Hopefully, your first Eclipse Workbench contains the Java perspective, which has four parts: (1) left column: the Package Explorer, which will list files and folders you are currently using, (2) a middle column, which will contain a Java file you are editing, (3) a right column, the Outline, which we don't explain, and (4) a lower row, which contains tabbed panes called Problems, Javadoc, and Declaration.

   Just above the Outline column, you should see the image shown to the right. Click the left icon to see perspectives that you can open (we advise against opening any in the beginning). The right icon indicates that the Java perspective is open. Right click that image to close the perspective or to reset it to its original. (Reset it if you managed to accidently move things around. For example, you might accidently close the left column.)

   Just above the left and middle columns is a toolbar, shown below. Some of the items are grayed out because they can't currently be used. Move your mouse slowly over the bar (in the Eclipse window); as it passes over an icon, a popup window will indicate what the icon is for. The left icon produces a dropdown menu for creating new things.

5. **Workspace**. A *Workspace* is a folder on your hard drive where Java programs are stored by Eclipse. You can have several workspaces, and you can decide where they should be on your hard drive. *However*, it is easiest in the beginning to have only one workspace and to let Eclipse decide where to place it. When Eclipse asks you to select a Workspace, use the one it wants to use, and make it the default.

6. **Project.** A (Java) *project* contains the Java code that you write, as well as related files needed to build, compile, and run the Java program. The project is actually a folder within the workspace. When you give the project a name, you are also giving a name to a new folder in your Workspace folder on your hard drive.

7. **Source files and machine language files**. A file that contains Java code has the suffix ".java". It is called a *source* file. The declaration of class Circle on the previous page would be in a file named Circe.java. When a source is compiled into the machine language, the resulting machine language file has the suffix ".class". For example, Circle.java is compiled into a file named Circle.class.

   Hint. In the project layout, create separate folders for source files and class files.

8. **Package**. A package is a set of Java classes that are grouped logically together and placed in a folder of their own on your hard drive. The name of the package is the name of the folder. In your first programs, your classes will be stored in the *default* package, but programming assignments may ask you to use other packages.

   Java comes with many packages for your use. For example, package java.lang contains classes for implementing Strings and a class Math that contains many math functions, package java.io contains classes for doing input/output, and package java.net has classes for implementing networking applications, like reading webpages.

9. **Constructing a new Project in Eclipse.** We now show you the steps in creating a new project with one Java class in it, class Circle given on the previous page. We talk about the effect of this on your hard drive.

   a. Make sure the Java perspective is showing. If not, use the "Open Perspective" icon to open it (see point 4).

   b. Construct a new project using menu item *File → New → Java Project* (or use an icon in the toolbar). Fill in the Project Name with the name CircleProject. We suggest creating separate folders for sources and class files. Click button *Finish* to set up the new Project with defaults.

      Hint. Always start a new project for a new Java program, e.g. for a new programming assignment. Putting files for different programs into one project creates a mess that will get you into trouble later.

   c. Construct a new class using men item *File → New → Class* (or use an icon in the toolbar). Give it the name Circle and click *Finish*.

   d. Look at the Eclipse window. You see a declaration for class Circle with nothing declared inside it. In the left column, you see CircleProject. If nothing shows below it, click the small arrow so that it points downard, and you should see src and JRE System Library. Click the arrow before src to get to see the default package, and click its arrow to see Circle.java in it.

      Now open a window on your hard drive and find where Eclipse stores the Workspace. In the Workspace you should see folder CircleProject. In it should be two folders: folder src contains Circle.java and folder bin contains Circle.class — Circle.java was compiled automatically to produce Circle.class.