AP COMPUTER SCIENCE
JAVA CONCEPTS I: LANGUAGE OVERVIEW

PAUL L. BAILEY

1. Data

1.1. **Classes and Objects.** Java is an object oriented language. This means that programs are organized around objects and their relationship to each other.

A *class* is a template for an object. A class is like the blueprint for a house: you can build a lot of houses with the same blueprint. In Java (like C#, but unlike C and C++), essentially all of the source code resides inside some class. An outline for the source code which defines a class looks like this:

```
public class ClassName extends ParentClassName
{
    // Fields

    // Constructors

    // Methods
}
```

Here, `ClassName` is the name of the class. The class `ClassName` *inherits* from class `ParentClassName`. This means that all of the fields and methods that are available `ParentClassName` are available for `ClassName`.

A *object* is an instance of a class. When an object of a given class is created, we say the object *instantiates* the class.

A *field* is an instance variable; this stores information related to the object. A *constructor* is a block of code that tells how to create an object of this class. A *method* is a labeled block of code (analogous to a *function* in C) which dictates the behavior of objects in this class.

Date: August 10, 2021.

1.2. **Variables.** *Variables* are names associated to data that is stored in the computer's memory.

1.2.1. *Kinds of Variables.* There are fours *kinds* of variables in Java:

- *Parameters* are passed to a method in its signature;
- *Local variables* are declared inside of a block of code inside of a method;
- *Instance variables* are declared inside a class, outside of a method, and are specific to an object;
- *Static variables* are declared inside a class, outside of a method, and are specific to the class.

1.2.2. *Categories of Variables.* There are two main *categories* of variables in Java:

- *Value variables* contain the value of the variable itself. These are also called *primitive variables* because they are variables which hold the primitive types `boolean` and the numeric types.
- *Reference variables* contain a reference to an object.

1.2.3. *Types of Variables.* There are several primitive *types* of variables in Java. The numeric types are the integral types and the floating-point types. The integral types are `byte`, `short`, `int`, and `long`, whose values are 8-bit, 16-bit, 32-bit and 64-bit signed two's-complement integers, respectively, and `char`, whose values are 16-bit unsigned integers representing UTF-16 code units (§3.1). The floating-point types are `float`, whose values include the 32-bit IEEE 754 floating-point numbers, and `double`, whose values include the 64-bit IEEE 754 floating-point numbers. The `boolean` type has exactly two values: `true` and `false`.

1.3. **Literals.** Literal values are also typed, but are not given a name and are wired into the code. For example:

- In `a = 1`, `a` is a variable and `1` is a literal integer.
- In `s = "This is a test"`, `s` is a variable and `"This is a test"` is a literal string.

2. Code

2.1. **Operators.** An *operator* takes one (unary), two (binary), or three (ternary) values and produces one value. Typically, in the code, an operator appears as one or two punctuation characters. The values are either variables or literals. For example:

```
!a      // NOT a       Logical unary operator
-a      // NEGATIVE a  Arithmetic unary operator
a+b     // a PLUS b    Arithmetic binary operator
a==b    // a EQUALS b  Relational binary operator
```

2.2. **Expressions.** An *expression* is a sequence of values joined by operators and parentheses. The expression is *evaluated*, which means it is reduced to one value. For example:

```
(a+b)*c         // Evaluated numerically depending on the types of a,b,c
(a+b)*c == 12   // Evaluated logically to either true or false
```

2.3. **Blocks.** A *code block* is a sequence of statements surrounded by braces. Statements are terminated by semicolons, or by code blocks. A *statement* is either a *declaration*, an *expression*, or a *flow control* statement. For example:

```
{                       // Begin block
    int i = 1;          // Declaration of a variable
    i = i + 2;          // Expression
    print(i);           // Method calls are considered expressions
    if (i > 3)          // Start of flow control statement
    {                   // Begin subblock within flow control statement
        i = 2;
    }                   // End subblock, and end of flow control statement
}                       // End block
```

2.4. **Methods.** A *method* is a code block with a label. A *method declaration* is the set of code in the program which declares the method. Method declarations have six components, in order:

- *Modifiers*, such as `public`, `private`, `static` and others.
- *Return type*, which is the data type of the value returned by the method, or `void` if the method does not return a value.
- *Method name*, which allows other parts of program to call the method.
- *Parameter list*, a comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses, (). If there are no parameters, you must use empty parentheses.
- *Exception list*, to be discussed later.
- *Method body*, a block of code which is executed when the method is called.

The *method signature* consists of the method name and parameter list. Methods with different signatures are different methods. In particular, it is allowable to use the same name more than once, as long as the parameter list is different. This is known as *method overloading*, and is a form of *polymorphism*.

The most common *access modifiers* are

- `public` - the field or method is accessible by any class
- `private` - the field or method is accessible only within the class in which it is defined
- `protected` - the field or method is accessible within the class or within any class which extends it, but not elsewhere

The `static` modifier states that the method is a *class method*, and does not require an instance of an object to be used. Lack of the `static` modifier indicates that the method is an *instance method*, and lives within a given object.

3. Examples

The Greenfoot IDE helps students build games. We built a game with crabs in a world which is modeled after Michael Kölling's *Joy Of Code* tutorial. See

`https://www.greenfoot.org/doc/joy-of-code`.

We illustrate these concepts with a sample of a `Crab` class.

```
public class Crab extends Animal
{
    private int score = 0;

    public void act()
    {
        move(3);
        stagger();
        ricochet();
        control();
    }

    public void addPoints(int points)
    {
        score += points;

        World world = getWorld();
        Counter counter = (Counter)world.getObjects(Counter.class).get(0);
        counter.setValue(score);
    }
}
```

In this case, the class `Crab` inherits from its parent class, `Animal`. This means that a Crab contains all the fields and methods of an Animal.

The `score` variable is a field of the `Crab` class.

The `act` and `addPoints` functions are instance methods of the `Crab` class.

Consider the line which says

`public void addPoints(int points);` .

The `addPoints` method has public accessibility and a void return type, which means it does not return anything. It take one parameter `points` of type `int`; this means that `points` may be used within the body of the method like any other variable. It is also possible for methods to have more than one parameter.

In the `addPoints` method, two local variables are declared; `world` of type `World`, and `counter` of type `Counter`. These are both reference types. Also in this method, the instance variable `score` is used. Note how the variables `score`, `points`, and `counter` are of three different types.

The question may arise whether or not parameters should be considered as local variables. Some references refer to parameters as a subcategory of local variables, and some refer to them as a separate category. In any case, local variables and parameters behave the same way in the body of the code of a method.

Department of Mathematics, Paragon Science Academy
Email address: pbailey@sonoranschools.org