Programming in C, Java, & More

Richard Robinson, B.Eng. Cand.

September 26, 2018

Contents

Chapter 1

Java Classes

1.1 Non-Static Classes

1.1.1 Format

A typical non-static class in Java has the following syntax and structure for the class file:

```
public class ClassName extends SuperClass implements Interface {

    private int paramOne, paramTwo;      // attributes section

    public ClassName() {}                // default constructer
    public ClassName(int paramOne) {}    // constructor with parameters

    public int getParam() {return 1;}    // accessor methods
    public void setParam(int param) {}   // mutator methods
}
```

In the attributes section, global attributes are declared; such attributes must be `private` and not initialized. A method or constructor can reference and distinguish these attributes via `this.param`.

1.1.2 Constructors

The constructors section typically is as follows:

```
public ClassName() { this(defaultValue); } // default
public ClassName(int paramOne) { this.setParam(paramOne); } // with parameters
```

The default constructor is the constructor with no arguments. Because of this, it is typically used to call another constructor that has parameters by passing a default value(s) as the parameter(s). Otherwise, constructors are generally used to initialize instances of the objects and to set attributes.

1.1.3 Methods

Methods are the most important aspect of classes, as they control what the class does. All classes should have both an accessor and mutator methods, with their formats typically being

```
public int getParam() { return this.param; } // accessor
public void setParam(int param) { this.param = param; } // mutator
```

The first method is an accessor, and is used to access the value of one of the private attributes of the class, which otherwise would not be visible to the client. As well, mutator methods change the value of the specified attribute to that of the parameter, and hence do not return any values.

1.1.4 Obligatory Methods

The built in `.equals()`, `.hashCode()`, and `.compareTo()` methods must manually be overwritten when creating a class. A typical `equals` override method to test for equality is as follows:

```
@Override
public boolean equals(Object obj) {
    if (obj != null && this.getClass() == obj.getClass()) {
        ClassName other = (ClassName) obj;
        return (this.param == other.param)
    } else return false;
}
```

A `compareTo` method is more customizable, having a default signature of

```
@Override
public int compareTo(ClassName name) { return 1; }
```

Lastly, the `hashCode` is simply implemented by calling the `cts.hash(params)` method.

1.1.5 Generics

Methods which support parameters of any Object type may be declared with the following syntax:

```
public <T> int methodName(T element) {}
```

In this case, `element` may be of any type and treated like a regular parameter. Moreover, to limit the types of objects the method supports, wildcards may be implemented in the signature to specify that a parameter must possess a certain quality.

For example, the following modification to the signature specifies the parameter's type must be one which implements the `Comparable` interface:

```
public <T extends Comparable<? super T>> int methodName(T element) {}
```

1.2 Aggregation

1.2.1 Intro to Aggregation

The concept of *aggregation* is a *has-a* relation such that one class is an instance of another. That is, `ParentClass` is said to have a `ChildClass`. In most cases, the parent's constructors and methods are given by

```
public Parent() { this(null, ...); }
public Parent(Child childName, ...) { this.setChildName(childName); }

public Child getChildName() { return this.childName; }
public void setChildName(Child childName) { this.childName = childName; }
```

Chapter 2

Bash Shell

Chapter 3

Intro to C

Chapter 4

The Processor

Chapter 5

Using RISC V

5.1 Start of Program

A typical RISC V or Assembly program starts with the following lines:

```
      96                # declares init address
      42, 100, 19, 2000   # stores values in mem
i     x1, x0, 96        # initializes init address
```

The first line contains the `ORG` command, which declares the initial address. The `DD` command stores its arguments in memory to be later used by the program. Lastly, the `addi` command with the `x1, x0` arguments initializes the registers to the initial address in `ORG`.

Values need not be explicitly stored in memory as will be discussed in the following section.

5.2 Using Registers

There are two primary methods of loading values into registers;

```
i     x4, x0, 42        # method 1
      x4, 8(x1)         # method 2
```

In the first method, the value 42 is directly loaded into `x4`. In the second method, the value must first be stored in memory via the `DD` command. To load the nth number of `DD`, `8n(x1)` is used as $8n + 96$ is the address of the value.

Consequently, values are loaded from registers into memory via

```
      x19, 88(x1)       # stores reg in mem
```

which stores the value contained in `x19` into memory address `88 + x1`, which in this example would be $88 + 96$.

5.3 Conditionals

In RISC V, *if* statements work unlike other languages, and operate using the following syntax:

```
  bne     x1, x2, Else        # goto Else
  # commands if true
  beq     x0, x0, Exit        # goto Exit
: # commands if false
:
```

This code snippet is equivalent to the more common format `if (x1 == x2) {} else {}`. The `beq` command goes to the label `Else` if `x1 == x2`. Consequently, the `bne` command does the reverse.

Chapter 6

Intro to Verilog

6.1 Syntax

A typical Verilog HDL program is formatted similarly to OOP languages, with a general program syntax given by:

```
ule mod_one(args); // initiates mod_one
/ commands
od_two name(args); // calls instance of mod_two
module

ule mod_two(args); // initiates mod_two
/ commands
module
```

In Verilog, values may be prefixed by optional base type and memory size. Variable values in Verilog use the following syntax when using all options:

```
fine VAR 12'h19c // has format <size>'<base><value>
```

If only the value is given, the default is decimal. As well, `?` may be used as a wildcard character in variables, for example `4'b10??`.

6.2 Logic

Verilog uses ternary instead of binary boolean logic. In addition to 0 (boolean false) and 1 (boolean true), there is also X (boolean unknown) and Z (boolean null) such that

$$(X \equiv \{0,1\}) \wedge 0 \implies Z \tag{6.1}$$

This leads to such logic having non-reflexivity as

$$0 \wedge 1 \implies 0 \quad \text{and} \quad 1 \wedge 1 \implies 1 \tag{6.2}$$