Recitation 3

Brandon Rozek
rozekb@rpi.edu

Rensselaer Polytechnic Institute, Troy, NY, USA

January 2022

Outline

Two things:

- Java
- Reasoning

Java

Compilers

- *Compilers* translates code written in a source language to a target language.
- Typically, the target language is considered "lower level" and machine dependent such as assembly, object, or machine code.
- Translating code to another higher-level target language is often called *transpilation*.
- Hence, Java is often considered a Hybrid language.

Interpreters

- An *interpreter* takes instructions written in a programming language and directly executes them.
- A language that does not require a compilation step is often called an *interpreted language*.

What does Java do?

- Java requires compilation to a target language called *Java bytecode*.
- The Java bytecode is then interpreted with the Java Virtual Machine (JVM)

Some Nuances...

- The JVM continously analyzes executions during runtime and decides to compile commonly executed paths using a technique called *just-in-time* (JIT) compilation.
- Note that compilation takes time, so there's often a balance!

The previous form of compilation where it happens before runtime is called *ahead-of-time* (AOT) compilation.

Subtype Polymorphism

Java supports *subtype polymorphism* which allows the programmer to use a subclass where a super class is expected.

```
Pet[] pets = new Pet[5];
pets[0] = new Cat("Pablo");
pets[1] = new Dog("Jackie");
```

Assuming that the classes Cat and Dog extend or are subtypes of Pet.

Static Binding

- Associating a name with a method or field is called *binding*.
- *Static binding* occurs at compile time and cannot be overridden.
- In Java, methods and fields that use the keywords *private*, *final*, or *static* are bound statically.

Dynamic Binding

- *Dynamic binding* associates names with methods or fields during runtime.
- Instead of the type information being used to decide which method to run, the object is inspected instead.

Overloaded Methods

- *Overloaded methods* is a language feature that allows for the same method name with different argument types.
- These are bound with statically.

```
class Animal {
    static void eat() {
        System.out.println("Animal␣is␣eating␣
            nothing...");
    }
    static void eat(Food f) {
        System.out.println("Animal␣is␣having␣a␣great␣
            meal!");
    }
}
```

Overridden Methods

- *Overridden methods* changes the method called when a subclass uses the same method name as the superclass.
- These methods are bound dynamically.
- Argument types must be the same but return type may differ.

```
public class Animal {
    void eat() {
        System.out.println("Animal␣not␣hungry.");
    }
}
class Dog extends Animal {
    @Override
    void eat() {
        System.out.println("Dog␣eat.");
    }
}
```

Question: What is the output?

```
class Animal {
    static void eat() {
        System.out.println("Animal␣eat.");
    }
}

class Dog extends Animal {
    public static void main(String args[]) {
        Animal a = new Dog();
        a.eat();
    }
    static void eat() {
        System.out.println("Dog␣Yum!");
    }
}
```

Dispatching

- Binding associates a name with a method/field.
- *Dispatching* determines which method to call given its arguments.

Reasoning through Code

Preconditions/Postconditions

- Precondition: Conditions that must hold before the code executes.
- Postcondition: Conditions that must hold after the code executes.

Forward/Backward Reasoning

- Forward Reasoning: Given a precondition, does a postcondition hold?
- Backward Reasoning: Given a postcondition, what is the precondition?

Forward Reasoning Example

Precondition: $\{x < -3 \;\&\&\; y == x\}$

```
x = x - 4;
y = x + abs(x);
z = (y + 5) * (x + 2);
```

What is the postcondition?

Backward Reasoning Example

What is the precondition?

```
t = 2 * s;
r = w + 4;
s = 2 * s + w;
```

Postcondition: $\{r > s \;\&\&\; s > t\}$

Practice: Forward Reasoning

Precondition: $\{\ s < 2\ \&\&\ w > 0\ \}$

```
t = 2 * s;
r = w + 4;
s = 2 * s + w;
```

What is the postcondition?

Practice: Backwards Reasoning

What is the precondition?

```
x = x - 4;
y = x + abs(x);
z = (y + 5) * (x + 2);
```

Postcondition: $\{x < -7 \;\&\&\; y == 0 \;\&\&\; z < -25\}$

Reasoning through If Statements

- Reasoning through if statements is similar to proof by cases.
- Requires keeping track of separate states of a program.

```
if (A) {
    // Postcondition B
} else {
    // Postcondition C
}
```

There are multiple ways to tackle it in order of its expressiveness:

- Keep track of it via implications. $\{A \implies B \;\&\&\; !A \implies C\}$
- Treat it as a disjunction. $\{B||C\}$
- Find commonalities between B and C

Example

What is the precondition?

```
if (x > 0) {
    x = x + 6;
} else {
    x = x / 2;
}
```

Postcondition: $\{|x| < 7\}$

Practice:

Precondition: $\{|x| > 5\}$

```
if (x > 0) {
    x = 3 - x;
}
else {
    x = x - 1;
}
```

What is the postcondition?

Reasoning through Loops

- A *loop invariant* is a property that is held at the beginning, after each iteration, and at the end of a loop.
- A good loop invariant should involve the loop variable and the postcondition.
- The negation of the loop condition (L_c) and the invariant (I) must imply the postcondition (P) at exit. $!L_c \;\&\&\; I \implies P$.
- We often prove loop invariants using *induction*.

Example:

```
// Precondition: a >= 0 && b >= 0
int mul(int a, int b) {
    int x = 0;
    int p = 0;
    while (p < b) {
        x = x + a;
        p = p + 1;
    }
    return x;
}
// Postcondition: x == a * b
```

Hoare Triples

- *Hoare Logic* is the formalization of reasoning through pre and post conditions.
- {`Pre`}`Code`{`Post`} is a succinct representation called a *Hoare triple*.

Weak vs Strong Conditions

- A condition Q is weaker than condition P if $P \implies Q$ but $Q \not\implies P$.
- We see this often with inequalities: $x < -5 \implies x < 0$

Any Questions?