Recitation 6

Brandon Rozek
rozekb@rpi.edu

Rensselaer Polytechnic Institute, Troy, NY, USA

February 2022

Outline

Two things:

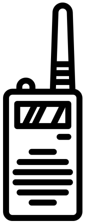
- ADT Methods
- Representation Invariants

Logistics

- First exam is next week.
- Let me know if you have any topics that you want me to go over.

Running Example

- I have an amateur radio license from the FCC, callsign KN4VYS.
- I am particularly interested in *software defined radio* (SDR)
- To make some of these concepts concrete, let's apply them to building an SDR!



Side Effects

- A method is said to have a *side effect* if it modifies some state outside its local context.
- Examples include: modifying object fields, printing to the screen, taking user input, network calls.

A program is not useful unless it has a side effect of some kind.

Reasoning through Side Effects

- Since side effects exist outside the method you're considering, this makes them difficult to reason through.
- The most common technique used to deal with side effects is through *immutability* and *exceptions*.

Personal Thoughts

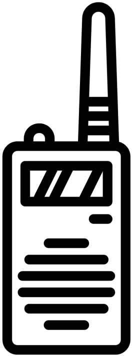
- The functional programming community have made great strides in dealing with side effects primarily through monadic structures. (Feel free to ask me!)
- Though a simple recommendation I have is to *decouple* computation from side effects.
- e.g A method `radiate()` should not also print to the screen

Reasoning through Abstract Data Types

- In fact, methods that are small and follow the *single-responsibility* principle are often easier to reason about and prove!
- At first, it'll feel verbose and repetitive. Later on you'll learn techniques which are more concise.

Example: Radio Overview

A radio (more specifically a transciever) contains both a receiver and a transmitter in one unit.



Receiver

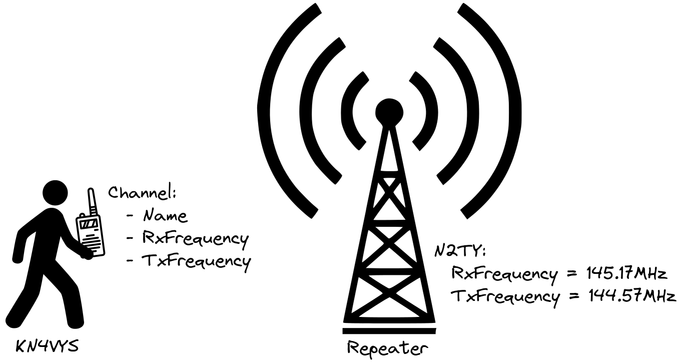
- Analog-to-Digital Conversion
- Demodulation
- Decode
- Play through Speakers

Transmitter

- Receive speaker input
- Encode
- Modulate
- Digital-to-Analog Conversion

Example: Connecting to a Repeater

Amateur radio operators extend their reach by connecting to a repeater. In the next slides, we'll analyze *channels*.



ADT Methods

We group the access methods of an ADT into:

- Creators: Creates a brand new object.
- Observers: Returns information about *this* object.
- Producers: Returns a new object of *this* type by performing operations on the current object.
- Mutators: Changes *this* objects fields.

Note: Immutable ADTs do not have mutators.

Creators

- Another name for constructors
- No preexisting state is changed, therefore `modifies` is none.

```
public Channel(double freq);
public Channel(double rxFreq, double txFreq);
```

Observers

- Returns information about *this* which often is of a different type (int, String, etc)
- Should have no *side effects*.

```
public bool isSimplex() {
    return this.rxFrequency == this.txFrequency;
}
```

Producers

- Returns an object of the same type, often with different fields.
- This technique is very common in immutable objects.
- Should have no *side effects*.

```
public Channel toDuplex(double txFrequency) {
    return new Channel(this.rxFrequency, txFrequecy)
}
```

Mutators

- Changes internal state of *this*
- Often does not return anything

```
public void changeRxFrequency(double frequency) {
    this.rxFrequency = frequency;
}
```

Immutability

- In immutable objects, instead of changing state through mutators, producers are used.

```
// Alternative to mutator
public Channel changeRxFrequency(double frequency) {
    return Channel(frequency, this.txFrequency);
}
```

Java strings are immutable and follow this pattern.

Immutability

- Big benefit to immutable objects is that they are thread safe by default!
- Slight performance detriment since memory gets copied.
- *For correctness, need to make sure memory is copied and not referenced in new object.*

Representation Invariant

Representation Invariant

- Similar to how we use loop invariants to reason about loops, we'll use *representation invariants* to reason about objects.
- Indicates whether data representation is *well-formed*.
- Must hold before and after every method.

Example: `this.frequency > 0`

Abstraction Function

- What does the data structure semantically mean?
- Example from class: array[2, 3, -1] represents $-x^2 + 3x + 2$

Representation Exposure

- Representation exposure is unintentional external access to the underlying representation of an object.
- Allows access without going through object's public methods
- Representation exposure can break representation invariants!

Example

Java does not provide any guarantees with the `private` keyword.

```java
public class Channel {
    private double rxFrequency;
    private double txFrequency;
    public Channel(double freq) {
        if (freq <= 0) {
            throw new IllegalArgumentException();
        }
        this.rxFrequency = freq;
        this.txFrequency = freq;
    }
    public static void main(String[] args) {
        Channel c = new Channel(14.54);
        c.rxFrequency = -5.0; // Bypassed our check :(
    }
}
```

Immutability Solution

With the `final` keyword, immutability is enforced.

```java
public class Channel {
    final private double rxFrequency;
    final private double txFrequency;
    public Channel(double freq) {
        if (freq <= 0) {
            throw new IllegalArgumentException();
        }
        this.rxFrequency = freq;
        this.txFrequency = freq;
    }
    // Add producer method setFreq
    public static void main(String[] args) {
        Channel c = new Channel(14.54);
        c.rxFrequency = -5.0; // Compiler error
    }
}
```

Beware of References

```
public class Channels {
    private ArrayList<Channel> cs;
    public Channels(ArrayList<Channel> cs_param) {
        this.cs = cs_param;
    }
    public Channels add(Channel c) {
        Channels new_channels = new Channels(this.cs);
        new_channels.cs.add(c);
        return new_channels;
    }
    public static void main(String[] args) {
        Channel n2ty = new Channel(145.17);
        Channels empty_channels = new Channels(new
            ArrayList<Channel>(););
        Channels some_channels = empty_channels.add(n2ty);
        System.out.println(empty_channels.cs.size());
    }
```

Recommendation: Copy

- When dealing with mutable producers, always make sure you perform a copy!
- Do note the difference between a shallow and a deep copy...

Defensive Programming

If you're dealing with mutable objects:

- Assume your invariants will be violated!
- Check the invariants on method entry, exit, etc.

Checking Invariants with Assertions

- Assertions can be used to check invariant satisfaction.
- java runs with assertions disabled by default (`java -ea` to enable)
- When assertions are not satisfied an `AssertionError` exception is thrown.

Example

```
public Channel toDuplex(double txFreq) {
    assert this.rxFrequency > 0;
    assert txFreq > 0;
    return new Channel(this.rxFrequency, txFreq);
}
```

Defensive Programming - Preconditions

- Recall that preconditions are what is assumed to be true before the method call.
- If its not true then all bets are off!
- But as we saw with *defensive programming* constraints will often get violated.

Preconditions and Exceptions

- This class recommends to have weaker preconditions in order to strengthen the specification.
- Make use of exceptions when inputs are invalid!

There might be some cases where weaker preconditions are undesirable...

Client Code: Try/Catch/Finally

The exception pattern is quite common with I/O operations.

```
public void mute_transmit() {
    Socket s;
    try { s = new Socket(host, port); /* ... */ }
    catch (ConnectionException e) { /* ... */ }
    finally { s.close(); }
}
```

Propagate Exceptions

In Java, *checked exceptions* must be either dealt or propogated by the caller. To propogate, use the `throws` keyword in the method declaration.

```
class Channels {
    /* Declarations from before... */
    public void loadConfigFromFile(String filename)
        throws IOException {
        FileReader file = new FileReader(filename);
        BufferedReader fileInput = new
            BufferedReader(file);
        fileInput.readLine(); // Might throw IOException
        fileInput.close();
    }
}
```

Any Questions?