

Week 13 - Friday

COMP 2000

Last time

- What did we talk about last time?
- Defect testing
- Validation testing
- Development testing
 - Unit testing
 - Component testing
 - System testing
- Release testing
- User testing
- Black box testing
- White box testing
- JUnit

Questions?

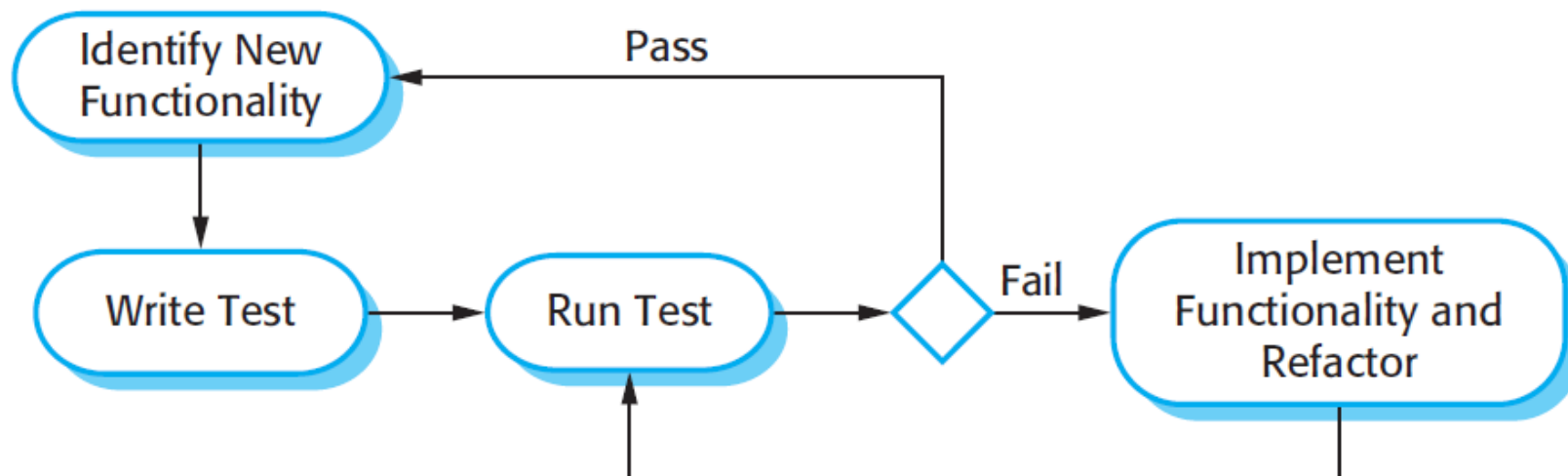
Project 4

Test driven development

- **Test driven development** (TDD) is an approach to development where testing and coding are interleaved
- Never move to the next increment of code until the current one passes its tests
- The key idea of TDD is that you write tests for the code **before** you write the code

TDD workflow

1. Identify the increment of functionality you want to add, typically small
2. Write an automated test for this functionality
3. Run the test (which should initially fail)
4. Implement the functionality and re-run the test, maybe fixing problems in existing code
5. Once all tests run successfully, move on to the next increment



Benefits of TDD

- By making the test first, you really understand what you're trying to implement
- Your testing has better code coverage, testing every segment of code at least once
- Regression testing happens naturally
- Debugging should be easier since you know where the problem likely is (the new code added)
- The tests are a form of documentation, showing what the code should and shouldn't do

More JUnit practice

Clock class

- Consider the following (partial) **Clock** class:

```
public class Clock {  
    private int hour;  
    private int minute;  
    private boolean am;  
  
    public Clock(int hour, int minute, boolean am) {  
        this.hour = hour;  
        this.minute = minute;  
        this.am = am;  
    }  
    public int getHour() { return hour; }  
    public int getMinute() { return minute; }  
    public boolean getAM() { return am; }
```

Additional Clock methods

- We want to finish the **Clock** class so that it implements the following methods

```
// Returns String representation of time, e.g. "09:47 AM"  
public String toString() {}  
  
// Adds a specific number of minutes to the current time  
public void addMinutes(int minutes) {}  
  
// Adds a specific number of hours to the current time  
public void addHours(int hours) {}  
}
```

Using test-driven development

- Instead of writing **toString()**, **addMinutes()**, and **addHours()** first, let's write a few JUnit tests for each of the three methods
- Then, we can come back and try to finish the methods so that they pass the tests

ArrayList class

- Consider the following (partial) **ArrayList** class:

```
public class ArrayList {  
    private int[] data = new int[10];  
    private int size = 0;  
  
    public int size() {  
        return size;  
    }  
}
```

- An **ArrayList** uses a dynamic array to store data (in this case, **int** values)
- Although there's initially room for 10 **int** values, none are stored yet (as indicated by **size**)
- After 10 **int** values have been added, adding another will require a larger array
- We double the length of the array (and copy over the old data) whenever it needs to be bigger

Additional ArrayList methods

- We want to finish the **ArrayList** class so that it implements the following methods

```
// Adds to end of list (determined by size)
// If there isn't enough space, double the length of the array
// and copy over the old data first.
public void add(int value) { }

// Get element at index
public int get(int index) {
    return 0; // Dummy return so that it compiles
}

// Set element at index
public void set(int index) { }

// Remove element at index (and return it)
public int remove(int index) {
    return 0; // Dummy return so that it compiles
}
}
```

Using test-driven development again

- Instead of writing `add()`, `get()`, `set()`, and `remove()` first, let's write a few JUnit tests for each of these methods
- Then, we can come back and try to finish the methods so that they pass the tests

Upcoming

Next time...

- Review up to Exam 1

Reminders

- **Work on Project 4**