COMP 201 – Software Engineering I
Lecture 29 – Implementation and Testing

Lecturer: Dr. T. Carroll

Email: Thomas.Carroll2@Liverpool.ac.uk
Office: G.14
See vital for all notes

Coming up…

Today

- Version Control
- Automated Testing
- Time for Office Hour Questions….

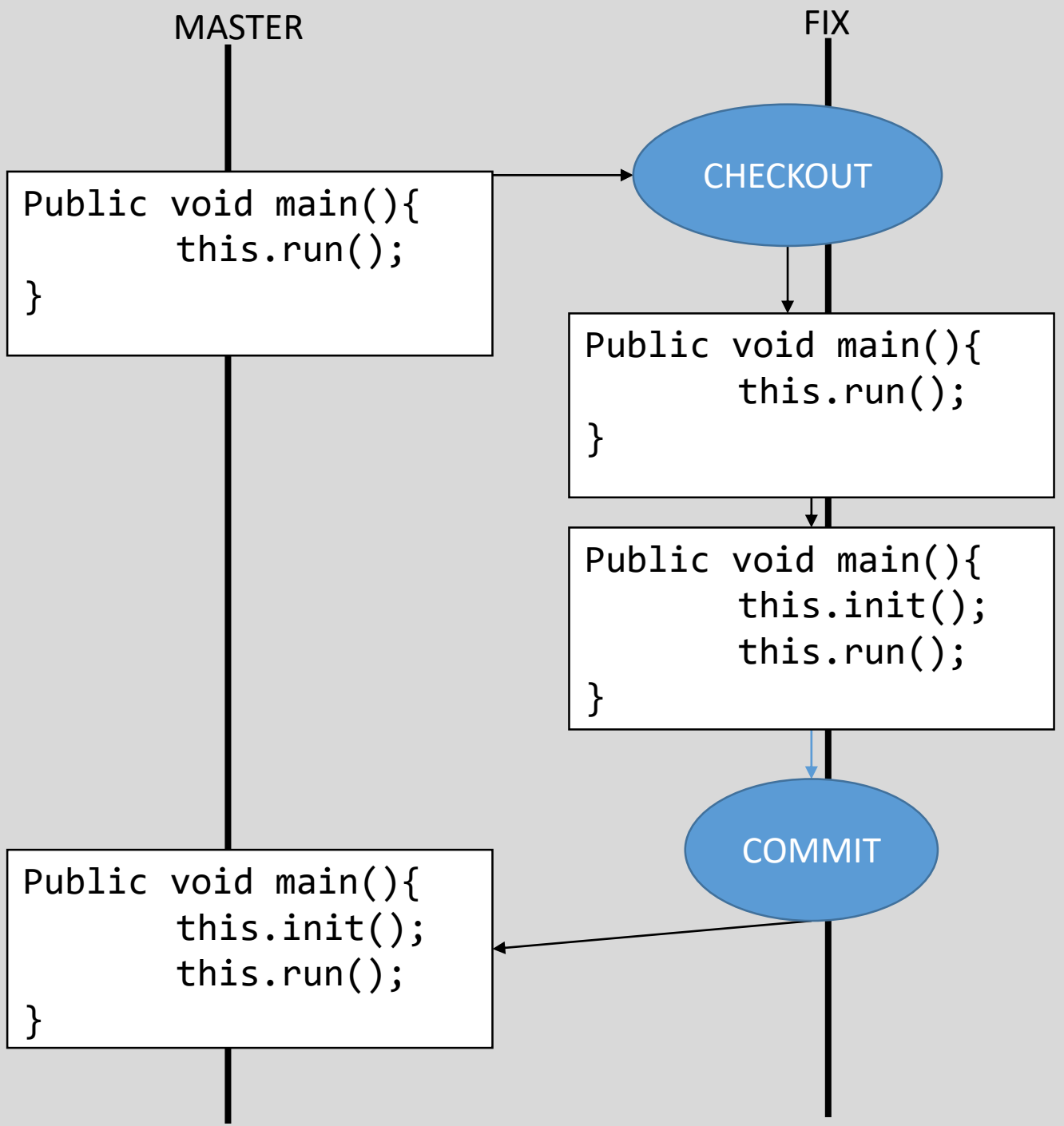
Version Control with Git

GIT – Version control can save the day!

- GIT is a widely used **version control system**
- Keeps **track of changes** made to the code – You can **commit** changes, and **revert** commits
- Create **branches** that isolate a bug fix or new feature
- **Merge** new changes into the master branch once tested
- Allows **multiple people** to work on the code
- Linux kernel is written in C and uses Git for version control
- GitHub is a popular online service that uses Git for code repositories and distribution

Branching

1. Error found in Master branch code
2. Checkout into Fix branch
3. Make and test the fix
4. Merge into Master branch



Exercise – A little time with GIT

- git init –bare : initialises a bare git repository
  git checkout <branch> : moves to a branch
  git add <file>: adds a file to be tracked by git
  git commit –m <message> : makes a commit with the message
  git push origin <branch> : Pushes changes to a branch
  git pull origin <branch> : pulls changes from a branch

Look on vital for the GIT instructions
Try to do this yourself
If not, please try to follow as I do them

Automated Testing with JUNIT

Automated Testing

- ***Any program without an automated test simply doesn't exist."*** *(Extreme Programming Explained, Kent Beck)*
- Software bugs have enormous costs :
  - time, money, frustration, and even lives.
- Creating and continuously executing test cases is
  - a practical and common approach to address software bugs.
- The **JUnit testing framework** is now the de facto standard unit testing API for Java development

What is JUNIT?

- JUNIT is an API that allows us to write **tests** that are conducted automatically
- Provides an **assertion** operation, to compare actual results against expected results.

Writing a test case (simplified version)

- Let us create a simple ***JUnit test case***, e.g. **`SimpleTest.java`**,
- Follow ***three simple steps***:

1. Import the necessary **JUnit** classes such as

   ```
   import static org.junit.Assert.*;
   import org.junit.Test;
   ```

2. Implement one or more *no-argument* **`void`** ***methods*** **`testXXX()`** ***prefixed by the word*** **`test`** and ***annotated*** as **`@Test`**
3. Implement these **`@Test`** methods by using **assertion** methods

Example

```
//import required JUnit4 classes:
import static org.junit.Assert.*;
import org.junit.Test;

public class SimpleTest
{
    @Test
    public void testSomething()
    {
        assertTrue("MULTIPLICATION???",4 == (2 * 2));
    }
}
```

Junit Running Example

```
C:\Antbook\ch04>javac -d build\test test\org\example\antbook\junit\SimpleTest.java

C:\Antbook\ch04>java
-cp build\test;C:\JAVA\junit4.8.2\junit-4.8.2.jar org.junit.runner.JUnitCore
org.example.antbook.junit.SimpleTest
JUnit version 4.8.2
.
Time: 0.01
OK (1 test)
```

Summary

- Git is a tool for version control
- Tracks changes to code as commits
- Allows creation of branches to isolate work-in-progress fixes or features (doesn’t break main code further!)
- Allows multiple people to work on code, manages the conflicts
- Junit is an API that lets us write test cases easily
- Allows us to compare expected with actual results
- Runs the tests with @Test automatically, displaying results
- Can also be used to create automatic reports for the tests