Object-Oriented Programming

Session: Week 5 Session 1
Instructor: Eric Pogue

Agenda:

1. Review this week's programming Assignment
2. Introduce the week's Learning Objectives
3. Topics & Examples

FastPrime… plus Questions

Write a performance optimized command line Java application that will programmatically find prime numbers [link] and store those numbers sorted in an output file.

In FastPrime we will create a command line Java application that will:

1. Use multiple threads to find the prime numbers between two numbers
2. Sort those results and store them to a file
3. Perform some timings
4. … And do this all very fast

See the details in this week's assignment

Application performance is ALWAYS a challenge. Learn how to optimize, test, and enhance the performance regularly. It can't be built in at the end of a project!

Learning Objectives – Week 5

1. Understand how Java uses **files** for input and output (IO)
2. Design and implement a controller class to **serialize** (reads & writes) data to a text file
3. Understand **threads** and how to develop and optimize multi-treaded Java applications
4. Understand Java **packages** and compile a class so that it belongs to a particular package
5. Import a class you write that belongs to a particular package.
6. Identify reasons for using JAR (Java ARchive) files to group together related java classes
7. Create a JAR file that stores the contents of a particular package
8. Explain software **testing** terms including unit, integration, user acceptance, performance testing, manual, automated, verification, validation, etc.
9. Understand the importance of testing and the criticality of finding/fixing defects early
10. Explain the purpose, syntax, and annotations of the various assert statements **JUnit** supports
11. Install JUnit onto your machine and execute a JUnit test on your application

Serialization and Writing/Reading Text Files (IO)

Serialization is an object-oriented programming term that means converting an object to a byte steam usually to be written to or read from a text or binary file.

To write to a text file:

- Create a File object, feeding the file's path to the File class constructor
- Create a FileWriter to access the File
- Create a BufferedWriter to write data to the FileWriter efficiently
- Use BufferedWriter's write and newLine functions to commit the data to the file.

To read from a text file:

- Create a File object, feeding the file's path to the File class constructor
- Attach a Scanner object to it
- Use Scanner's readLine and hasNextLine functions to read the file

A buffered writer or buffered stream efficiently organization reads and writes for optimal disk performance. The danger is that if there is a power failure (or someone accidentally kicks your power strip) you could lose data. It is good to "flush the buffer" at critical times when using a buffered writer on mission critical projects. There is little or no danger in using a buffered reader.

Binary files versus text files.

XML – Text file format for structured data… HTML for data.
JSON – "Simplified" text file format for structured data.
Binary

XML Example

In computing, XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is an open standard that:

- Supports nearly all development languages and platforms
- Allows us to cross between many applications
- Can result in large files
- Supports schema to validate data

```xml
<?xml version="1.0" encoding="UTF-8"?>

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

JSON Example

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is an open standard that:

- Supports nearly all development languages and platforms
- Allows us to cross between many applications
- Can result in large files

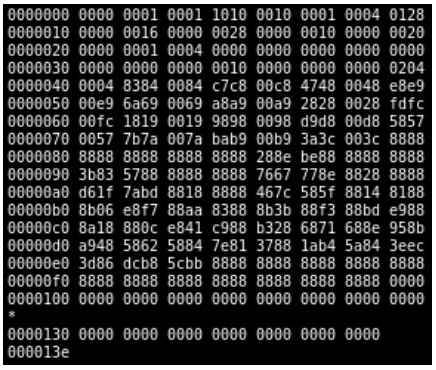
```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

JSON is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is a very common data format used for asynchronous browser/server communication, including as a replacement for XML in some web service style systems.

Binary Files

A binary file is a computer file that is not a text file. The term "binary file" is often used as a term meaning "non-text file". The can be open or closed formats that are generally:

- Fast, small, and efficient*
- Often not very portable across applications and platforms
- Difficult to maintain backward compatibility



Some people would say that binary files include all files, and that text files are just binary files that are being interpreted in a specific way.

Serialization Example

We are going to demonstrate Writing and Reading OvalDraw data to a proprietary text file by:

- Starting with the OvalDraw application
- Hooking up the “Open” and “Save” menu events
- Teaching our OvalDraw class to Write and Read itself
- Saving our session
- Loading our session

A buffered writer or buffered stream efficiently organization reads and writes for optimal disk performance. The danger is that if there is a power failure (or someone accidentally kicks your power strip) you could lose data. It is good to “flush the buffer” at critical times when using a buffered writer on mission critical projects. There is little or no danger in using a buffered reader.

Binary files versus text files.

XML – Text file format for structured data… HTML for data.
JSON – “Simplified” text file format for structured data.
Binary

End of Session

Course Number: CPSC-24500

Week: 5

Session: 1

Instructor: Eric Pogue

Object-Oriented Programming

Session: Week 5 Session 2
Instructor: Eric Pogue

Agenda:

1. Briefly review FastPrime assignment
2. Review the week's Learning Objectives and themes for the week
3. Performance Optimization
4. Threads and more Threads… and more Threads

Get ready next week for some Visual Studio and C# work.

FastPrime… plus Questions

Write a performance optimized command line Java application that will programmatically find prime numbers and store the numbers sorted in an output file.

In FastPrime we will create a command line Java application that will:

1. Use multiple threads to find the prime numbers between two numbers
2. Sort those results and store them to a file
3. Perform some timings
4. … And do this all very fast
5. Come to our Thursday lunch session with any questions… or email your question head of time

See the details in this week's assignment.

Application performance is ALWAYS a challenge. Learn how to optimize, test, and enhance the performance regularly. It can't be built in at the end of a project!

Note that I have made updates to this week's assignment. Please be sure to get the current version (it says "version 2" at the top) before you complete the assignment.

Learning Objectives – Week 5

1. Understand how Java uses **files** for input and output (IO)
2. Design and implement a controller class to **serialize** (reads & writes) data to a text file
3. Understand **performance optimization, threads,** and how to develop and optimize multi-treaded Java applications
4. Understand Java **packages** and compile a class so that it belongs to a particular package
5. Import a class you write that belongs to a particular package.
6. Identify reasons for using JAR (Java ARchive) files to group together related java classes
7. Create a JAR file that stores the contents of a particular package
8. Explain software **testing** terms including unit, integration, user acceptance, performance testing, manual, automated, verification, validation, etc.
9. Understand the importance of testing and the criticality of finding/fixing defects early
10. Explain the purpose, syntax, and annotations of the various assert statements **JUnit** supports
11. Install JUnit onto your machine and execute a JUnit test on your application

Themes for the week:
- Writing/Reading files and Serialization
- XML & JSON
- **Performance, performance, performance…**
- **Optimization through threads**

Note that given the industry prioritization on developing high performance multithreaded applications, I have made multithreading a higher priority than in past terms of this class… this has resulted in JUnit and automated unit testing becoming a bit less of a focus.

- Effectively packaging Java class files for sharing
- Delivering quality products through **design, development, and testing**

Performance Optimization and Threading

Performance is critical in application development… the focus of performance optimization continues to evolve, but the criticality remains very high! Multithreading is one very important way that we can optimize CPU performance; however, there are many other performance bottlenecks and optimization techniques:

- CPU… threading
- Memory… optimize disk usage, buy more memory
- Disk IO… buffering, file size, or faster (more expensive) disks
- Network bandwidth… "file" or package size
- Network latency… pray for a miracle!
- User Interaction and Capabilities

I believe that performance optimization is one of the most important and challenging aspects of developing high quality software.

Consider mobile phone networks and satellite networks… and latency.

User experience.

Know the difference between latency and bandwidth and how it impacts network and application performance:
https://en.wikipedia.org/wiki/Network_performance

The following measures are often considered important:
Bandwidth commonly measured in bits/second is the maximum rate that information can be transferred
Latency the delay between the sender and the receiver decoding it, this is mainly a function of the signals travel time, and processing time at any nodes the information traverses
Throughput is the actual rate that information is transferred
Jitter variation in packet delay at the receiver of the information
Error rate the number of corrupted bits expressed as a percentage or fraction of the total sent

You can usually buy more bandwidth. Fixing a latency issue might require you to change the speed of light.

Threads & Multithreaded Applications

Multithreading: A technique by which a single set of code can be used by several processors or cores at different stages of execution.

- Threads and application performance are becoming nearly synonymous
- Moore's law only remains achievable if we can effectively utilize multi-processor, multi-core, and multi-threaded applications
- Our performance principles that we discuss will be applicable across platforms and environments
- Our practical focus will be on Java multi-threading
- Parallel processing has become the focus of the computing and software development industry
- The rise of big data, artificial intelligence, virtual/augmented reality, and dedicated graphical processing units (GPUs) have made that a nearly guaranteed trend for years to come

The difference between user perceived (real or perceived) is as important as actual performance. Optimizing application performance to reflect user capabilities is challenging and necessary. It doesn't matter if you optimize an application to be 10 times faster if the bottleneck is how fast the user.

Multithreading is the primary method of optimizing CPU performance. It is unlikely to be of benefit if the bottleneck is elsewhere. For example, if you are disk bound, splitting your process into multiple threads is unlike to be of benefit.

Processors, Cores, and Threads

- Computers have one or more Processors (CPUs)
- Processors each have one or more Cores
- Cores can create one or more Threads
- An application running only on one thread of a dual cpu, quad-core, single thread can utilize only a portion of 1/8th of the processing power of that machine

For this discussion we will using multithreading and multiprocessing terms synonymously for our purposes.

Modern central processing units (CPUs) are made up of cores. A core is like a mini-processor that works with its fellow cores to perform the work that applications request of the CPU. In the old days, a CPU had just one core, a single channel through which all requests would pass. This was how we optimized applications… CPU, memory, fast disk, slow disk. Today, though, with multiple CPUs and multiple cores, a CPU can pay attention to and do many things at once.

This architecture, in turn, allows today's applications to perform multiple tasks at once. This ability is called multitasking. Multitasking enables an

Multi-Threaded Development

Now for the bad news. Multi-Threaded Development is really hard!

Image sources: Google

- Rigidity - It is hard to change because every change affects too many other parts of the system
- Fragility - When you make a change, unexpected parts of the system break
- Immobility - It is hard to reuse in another application because it cannot be disentangled from the current application

Stadia add-in net change example.

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlock occurs when multiple threads need the same locks but obtain them in different order.

Multi-Threaded Development

Now for the bad news. Multi-Threaded Development is really hard!

- Developing commercial quality multi-threaded applications makes Rigidity, Fragility, and Immobility much harder to avoid
- Many of the 3rd party professional libraries that the industry had come to rely on came into question as multi-threading application became required
- Testing becomes harder when a sequence of events becomes variable
- What if your automated unit test results might be different depending on which thread finishes first?
- What about deadlock?
- **Performance** is so important that we will need to understand be able to effectively utilize, test, and deploy effective multi-threaded applications

- Rigidity - It is hard to change because every change affects too many other parts of the system
- Fragility - When you make a change, unexpected parts of the system break
- Immobility - It is hard to reuse in another application because it cannot be disentangled from the current application

Increased complexity is the primary disadvantage for developing multithreaded applications.

Some languages have come into existence in order to try to reduce the complexity of writing, enhancing, and supporting multithreaded applications. For example, Scala has implemented specific parallelization features in the core language that make it a first class threading language. Note that Scala also targets the Java runtime environment.

C++ would be an example of a language that has implemented a plethora of threading mechanisms for various platforms and implementations. Recent versions have introduced more common approaches.

Stadia add-in example.

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlock occurs when multiple threads need the same locks but obtain them in different order.

Interesting threading article:

http://blog.smartbear.com/programming/why-johnny-cant-write-multithreaded-programs/

When reviewing libraries look for something like "This class is immutable and thread-safe." before you use it in multithreaded development.

Multi-Threaded Example

ThreadedRandomNumbers: Calculate 1,000,000,000 random numbers between 1 and 2,000,000. Print “We found number 1024!” to the console each time 1024 is generated. We would expect it to come up approximately 500 times.

- Write a single threaded application
- Divide the application into multiple threads and repeat
- Consider the diminishing returns of adding additional threads
- Implement by extending Thread
- … and by implementing Runnable

It’s good to have a trivial subject matter when starting with threaded applications. It’s challenging enough to learn a complex new topic when working with a random number or a prime number.

Bonus 10 points for anyone who can implement this application using “implements Runnable” and have it execute at the same speed as the “extending Thread” version… or explain why it performs consistently slower than the extends Thread implementation.

End of Session

Course Number: CPSC-24500

Week: 5

Session: 2

Instructor: Eric Pogue

Object-Oriented Programming

Session: Week 5 Session 3
Instructor: Eric Pogue

Agenda:

1. Discuss FastPrime… bring your questions
2. Other topics as time allows

FastPrime… plus Questions

Write a performance optimized command line Java application that will programmatically find prime numbers [link] and store those numbers sorted in an output file.

In FastPrime we will create a command line Java application that will:

1. Use multiple threads to find the prime numbers between two numbers
2. Sort those results and store them to a file
3. Perform some timings
4. … And do this all very fast

See the details in this week's assignment

Application performance is ALWAYS a challenge. Learn how to optimize, test, and enhance the performance regularly. It can't be built in at the end of a project!

Given our experience with ThreadedRandomNumbers you may want to experiment with extending Thread vs. implementing Runnable?

End of Session

Course Number: CPSC-24500

Week: 5

Session: 3

Instructor: Eric Pogue

Object-Oriented Programming

Session: Week 5 Session 4
Instructor: Eric Pogue

Agenda:

1. Finish Multi-Threading Example… implement Runnable
2. Java Packages & JAR Files
3. Software Testing and JUnit

Multi-Threaded Example

ThreadedRandomNumbers: Calculate 1,000,000,000 random numbers between 1 and 2,000,000. Print "We found number 1024!" to the console each time 1024 is generated. We would expect it to come up approximately 500 times.

- Write a single threaded application
- Divide the application into multiple threads and repeat
- Consider the diminishing returns of adding additional threads
- Implement by extending Thread
- … and by implementing Runnable

It's good to have a trivial subject matter when starting with threaded applications. It's challenging enough to learn a complex new topic when working with a random number or a prime number.

Bonus 10 points for anyone who can implement this application using "implements Runnable" and have it execute at the same speed as the "extending Thread" version… or explain why it performs consistently slower than the extends Thread implementation.

Learning Objectives – Week 5

1. Understand how Java uses **files** for input and output (IO)
2. Design and implement a controller class to **serialize** (reads & writes) data to a text file
3. Understand **threads** and how to develop and optimize multi-treaded Java applications
4. Understand Java **packages** and compile a class so that it belongs to a particular package
5. Import a class you write that belongs to a particular package.
6. Identify reasons for using JAR (Java ARchive) files to group together related java classes
7. Create a JAR file that stores the contents of a particular package
8. Explain software **testing** terms including unit, integration, user acceptance, performance testing, manual, automated, verification, validation, etc.
9. Understand the importance of testing and the criticality of finding/fixing defects early
10. Explain the purpose, syntax, and annotations of the various assert statements **JUnit** supports
11. Install JUnit onto your machine and execute a JUnit test on your application

Java Packages

A Java **package** is a group of related classes. We have been dealing with packages throughout this course. The most fundamental aspects of the language are held in java.lang, a package you get for "free" without having to import. We have also regularly used packages like javax.swing and java.awt.

Advantages of packages include:

- Eliminates naming conflicts
- Stores related classes together
- Optimizes class access in a package… think of a related hierarchy in a package

Java Package Creation

A Java **package** is created by including a "package" statement at the top of a Java file along with a unique identifier.

Package creation standards generally include:

- Utilizing an owned domain name in reverse to guarantee uniqueness
- Using "edu.lewisu.cs.24500shapeslibrary" for ShapesLibrary would be a good example
- Utilizing JAR files to distribute and manage Java class files

When you do that, the .class file for what you are declaring is supposed to be put in the folder that corresponds to that package. In this case, it will be placed in edu/lewisu/cs/24500shapeslibrary/
To make sure that that happens, you want to compile it as follows
javac –d . ShapesLibrary.java

The "-d ." part indicates that the current directory is the root, and the package folders will be created underneath that root. If a class is supposed to belong to a package, you must include the –d compiler directive.

So, in this example, the .class file will be placed in
<current_directory>/edu/lewisu/cs/24500shapeslibrary/

One of the nice things about doing it this way is that you can distribute your .class files easily without revealing your source files.

JAR Files and Java Packages

A JAR (Java ARchive) file [link] is a Java Package file format used to distribute Java class files.

JAR files are:

- Used to deploy full applications or significant components
- Usually compressed to optimize download times and storage space
- Utilized to deploy much more than just Java class files
- Optionally digitally signed to provide security… we can know who the package came from and that it is unaltered
- Optionally versioned to provide for effective updating

Several related file formats build on the JAR format:

- WAR (Web application archive) files, also Java archives, store XML files, Java classes, JavaServer Pages and other objects for Web Applications.
- RAR (resource adapter archive) files (not to be confused with the RAR file format), also Java archives, store XML files, Java classes and other objects for J2EE Connector Architecture (JCA) applications.
- EAR (enterprise archive) files provide composite Java archives that combine XML files, Java classes and other objects including JAR, WAR and RAR Java archive files for Enterprise Applications.

To create a jar:
jar cf nameofjar.jar list of files most of which are class files

To extract a jar:
jar xf nameofjar.jar

The jar file can contain an entire directory, and even a hierarchy of subdirectories.

So, for example, you can build a jar file for our edu.lewisu.cs.shapes package:
jar cf shapes.jar edu

Java Classpath

Classpath [link] is a parameter in the Java runtime (java) or the Java compiler (javac) that specifies the location of user-defined classes and packages. The parameter may be set either on the command-line, or through an environment variable.

The Classpath:

- Allows us to efficiently find compilation and runtime dependencies
- Provides visibility to third part or inhouse libraries (JAR files)
- Allow us to compile or execute with different dependencies by simply resetting the Classpath parameter
- Can be set “permanently” through an environment variable or temporarily on the command line

javac –cp .:./shapes.jar ShapeTest.java
javac –cp .:./shapes.jar ShapeTest

The “-cp” stands for classpath, and it allows me to specify where classes should be grabbed from. The “.” means “current directory”. The “:” separates different parts of the classpath. The statement above means that the classpath includes the current directory and the shapes.jar file that is in the current directory. This will work on a mac or linux machine.

Note that, on Windows, the class path would be specified as .;.\shapes.jar. Again “.” means “current directory”, but, on Windows, the “;” is used to separate different pieces of the path instead of a colon, and a backslash is used to separate folder names instead of a forward slash.

Package, JAR, and ClassPath Example

Let's make ShapesLibrary a "real" library. We will:

- Make ShapesLibrary a Package
- Place our Package in a JAR
- Test our JAR using ShapeDraw
- Set up a ClassPath to access our ShapesLibrary from ShapeDraw

Command line syntax
javac -d . Circle.java
jar cf shapes.jar edu
javac -cp shapes.jar CircleTest.java
java CircleTest

Add Package command to library:

package com.epogue.sl;

Compile ShapesLibrary.java:

javac -d . ShapesLibrary.java

Create JAR file:

jar cfv ShapesLibrary.jar com

Display contents of JAR file:

jar tf .\edu.lewisu.cs.24500shapeslibrary.jar

Package, JAR, and ClassPath Example

Create a Circle library and use it in a CircleTest application including:

- Create Circle.java and CircleTest.java
- Compile Circle.java as a Package
- Place the Circle Package in a JAR file
- Compile CircleTest against the Circle JAR file
- Run CircleTest

Command line syntax
javac -d . Circle.java
jar cf shapes.jar edu
javac -cp shapes.jar CircleTest.java
java CircleTest

Add Package command to library:

package com.epogue.sl;

Compile ShapesLibrary.java:

javac -d . ShapesLibrary.java

Create JAR file:

jar cfv ShapesLibrary.jar com

Display contents of JAR file:

jar tf .\edu.lewisu.cs.24500shapeslibrary.jar

Learning Objectives – Week 5

1. Understand how Java uses **files** for input and output (IO)
2. Design and implement a controller class to **serialize** (reads & writes) data to a text file
3. Understand **threads** and how to develop and optimize multi-treaded Java applications
4. Understand Java **packages** and compile a class so that it belongs to a particular package
5. Import a class you write that belongs to a particular package.
6. Identify reasons for using JAR (Java ARchive) files to group together related java classes
7. Create a JAR file that stores the contents of a particular package
8. Explain software **testing** terms including unit, integration, user acceptance, performance testing, manual, automated, verification, validation, etc.
9. Understand the importance of testing and the criticality of finding/fixing defects early
10. Explain the purpose, syntax, and annotations of the various assert statements **JUnit** supports
11. Install JUnit onto your machine and execute a JUnit test on your application

Software Testing Overview

Software Testing [link] is important because, if done right, it can help us find and fix problems earlier and make our system delivery process less immobile, rigid, and fragile.

As future developers in an object-oriented programming class, we are going to:

- Understand testing within the various Software Development Lifecycles
- Understand testing terminology
- Know how to develop applications that are easier to test
- Master Unit Testing and Automated Testing

Never underestimate the value of good design and implementation (for testability, encapsulation, etc.) on the economics of testing. You can't afford to test in quality!
Developers are responsible for product quality. Tester should be able to minimize that chance that a defect makes it to production.

We test to find defects and/or to validate that we have not introduced new defects.

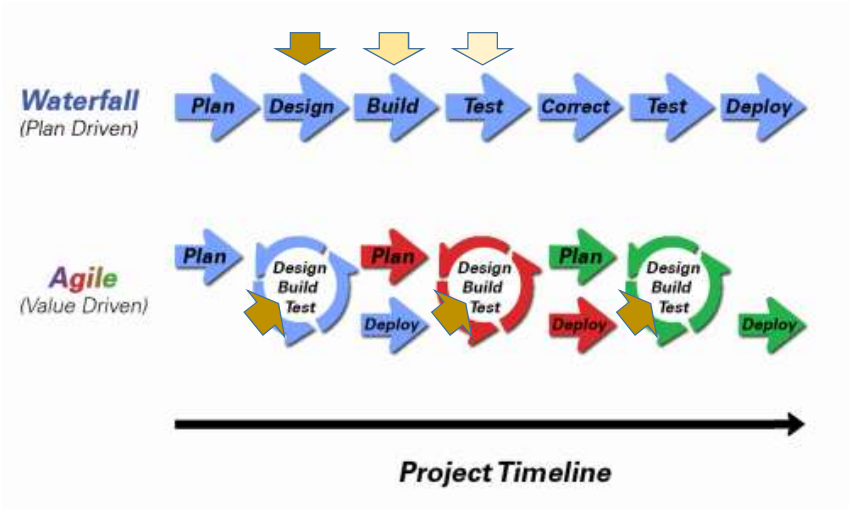
Defects are exponentially more expensive to fix the longer the exist.
Performance issues are often the most difficult and expensive defects to fix. They are often not found until the application if running under production load… which is often only when it is in production.

Unit - $200
Integration - $600
User Acceptance - $6,000
Production - $100,000+

The permutations of modern software features, data, tools, environments, etc. quickly becomes unmanageable. Testability needs to be goal of nearly all non-trivial applications.

Dave Cutler of Windows NT fame had a quote. I wish I could remember the exact words, but it went something like, "I hate having testers because they give developers the false hope that someone else can save them from their sins."

Object-Oriented Programming within Various Development Methodologies



Development Methodology and Software Development Lifecycle (SDLC) are often used interchangeably.

The Iterative development methodology is not depicted here as even the mainstays and inventors of the Iterative development methodology seem to be moving toward agile. Plus as Waterfall "holdouts" move, they seem to be moving directly toward Agile. Can you start to see my biases?

Development Methodologies (SDLCs) are a future Bonus Topic. There are several optional slides t the end of this deck. Let me know if you would like to have a more formal overview of the topic as part of this class. I have a passion in this area.

Object oriented-programming concepts/practices evolve and reprioritize depending on the development methodology.

For example, in Waterfall (as well as in Iterative) object-oriented design often play a critical role in the (big upfront) design activities. UML diagrams and project artifacts are often important to the overall project success. (opinion) Practical reality has been that these design artifacts often do not reflect the actual implementation and are rarely maintained or updated.

The Agile practitioners do not reject these design artifacts. However, the focus on shorter time horizons, evolving architecture, and working code changes the value proposition for object-oriented practices to more focus on the build, test, enhance activities.

Waterfall vs Iterative vs Agile

	Waterfall	Iterative	Agile
References	United States Department of Defense: DOD-STD-2167A (1985)	Rational Unified Process (RUP) Open Unified Process	Scrum Kanban Scaled Agile Framework (SAFe)
Priorities	Planning and predictability	Architecture, modeling, and efficiency through early detection & fixing of issues	Responsiveness to feedback, efficiency through engineering practices, early detection & fixing of issues
Principles	Execute phases sequentially: 1. Requirements 2. Analysis 3. Design 4. Coding 5. Testing 6. and Operations Define and commit to Scope, Cost, and Timeline "early" Implement strict Change Control	Develop and test iteratively Manage requirements Use components Model visually Verify quality Control changes	Develop, test, deploy, and release iteratively Capture lightweight near term requirements Empower teams Allow requirements to evolve but maintain fixed timelines Apply engineering practices and systems thinking (e.g. TDD) Integrate early user feedback into remaining plan Maintain a collaborative approach between all stakeholders

The views on testing and change management have changed dramatically as organizations have moved from Waterfall to Iterative to Agile development lifecycles.

Successful Agile (and Iterative) Development REQUIRES better application design, development techniques, and testing practices.

Testing and lack of defects are not the end goal. A higher quality more usable more cost effective product is the goal.

Testing Terms

Important testing terms include:

- Unit Testing: developer testing their own code
- Integration Testing: development team testing their full code
- System Testing: multiple development teams testing a full system or systems
- Performance Testing: testing performance at the Unit, Integration, and/or System level
- Manual Testing: a person using the application often running test scenarios
- Automated Testing: a group of automated tests that run on the application in the Unit, Integration, System, or Performance testing areas
  - UI Automated Testing attempts to exercise the application be reproducing user events (key & mouse events)
  - API Automated Testing occurs at the function/method API level (JUnit for example)

Testing Terms

Important testing terms (continued):

- Verification: does the application perform as expected
- Validation: does the application provide the business benefit that was expected
- Behavioral Testing: verifying that the correct functions were called with the correct parameters
- State Testing: focuses on the results of those calls

Automated Testing

Automated testing has become a more important part of effective software testing. The pros of Automated Testing include:

- Repeatable tests that are quick to run and can support Iterative and Agile development
- Very effective in validating environments and doing “smoke tests” to make sure a new build meets a minimal set of requirements
- Supports Performance Testing very effectively
- Finds range defects
- Very inexpensive and quick to repeat testing and validate fixes

False positives versus valid defects found.

Manual Testing and Automated Testing can be supportive of each other.

Automated Testing

Some of the benefits of Automated Testing have been oversold. Some of the challenges include:

- Developers only rarely can come up with scenarios in scripts that they would not already have tested in their unit testing… they often don't know what they don't know
- UI focused Automated Testing (key & mouse events) are often challenging and create a great number of false-positives
- Automated Testing investment is often not prioritized or tracked so it is difficult to know its effectiveness
- Scripts can be expensive to develop and maintain

False positives versus valid defects found.

Manual Testing and Automated Testing can be supportive of each other.

JUnit

JUnit [link] is a Automated Testing framework focused on developing and running Unit test for Java applications. JUnit:

- Is a Java opensource extension
- Provides annotations to identify test methods.
- Provides assertions for testing expected results.
- Provides test runners for running tests.
- JUnit tests allow you to write codes faster, which increases quality.
- JUnit is elegantly simple. It is less complex and takes less time.
- JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results

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

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

@Test
identifies the method as a test method (remember, method and function are synonyms)

@Test(expected = Exception.class)
fails if the method does not throw the named exception

@Test(timeout=100)
fails if the method takes longer than 100 milliseconds

@Before
public void method()
This method is executed before each test. It is used to prepare the test environment.

@After
public void method()
This method is executed after each test. It is used to clean up the test environment, including cleaning up expensive memory structures.

@BeforeClass
public static void method()
This method is executed once, before any test is done. It is used to do time-intensive tasks before any test is done.

End of Session

Course Number: CPSC-24500

Week: 5

Session: 4

Instructor: Eric Pogue

JUnit Notes (continued):
@AfterClass
public static void method()
This method is excuted once after all tests have finished.

Include in your functions various assert statements. The JUnit Assert class has several static methods that throw exceptions (specifically, AssertionException objects) when the assert you are testing fails.
assertTrue(boolean condition)
assertFalse(boolean condition)
assertEquals(expected, actual)
assertEquals(expected, actual, tolerance), where tolerance is the number of decimals that must be the same
assertArrayEquals(expected, actual)
assertNull(object) checks that the object is null
assertNotNull(object)
assertSame(expected, actual) – checks to see if they correspond to the same object
assertNotSame(expected, actual)
assertThat(object, matcher), where you can write your own matcher class to test some more complicated condition you want to assert. (see http://tutorials.jenkov.com/java-unit-testing/matchers.html, for example)

Running JUnit Test from the command line:
With the classpath set as shown previously, compile all the functions:
javac SoftwareUnderTest.java UnitTests.java TestRunner.java

Then run TestRunner:

```
java TestRunner
```