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

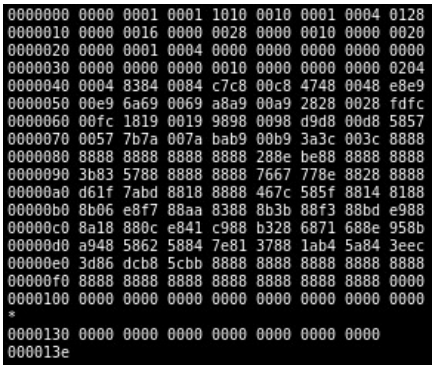
```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "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. Very briefly review FastPrime assignment
2. Review the week's Learning Objectives
3. Performance optimization
4. Threads and more Threads… and more Threads

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 **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

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

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 [link] 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 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

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!

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

Testing

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.

Integration (or Functional) Test: Another kind of test. These test whether the functionality of the entire application works as it should, or that the entire application behaves as expected.

Performance Test: Still another kind of test. These benchmark the performance of an application, especially under high load.

Behavior vs. State Testing: Behavior testing focuses on verifying that the correct functions were called with the correct parameters. In other words, are the various pieces interacting correctly? State testing focuses on the results of those calls. We will focus on state testing.

Manual Testing	Automated Testing
Executing a test cases manually without any tool support is known as manual testing.	Taking tool support and executing the test cases by using an automation tool is known as automation testing.
Time-consuming and tedious − Since test cases are executed by human resources, it is very slow and tedious.	**Fast** − Automation runs test cases significantly faster than human resources.
Huge investment in human resources − As test cases need to be executed manually, more testers are required in manual testing.	**Less investment in human resources** − Test cases are executed using automation tools, so less number of testers are required in automation testing.
Less reliable − Manual testing is less reliable, as it has to account for human errors.	**More reliable** − Automation tests are precise and reliable.
Non-programmable − No programming can be done to write sophisticated tests to fetch hidden information.	**Programmable** − Testers can program sophisticated tests to bring out hidden information.

False positives versus valid defects found.

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