OO ANALYSIS & DESIGN

CPSC 240 Course Pack

Abstract

This document contains material for Gusty's CPSC 240. Gusty took Dr. Karen Anewalt's CPSC 240 in Spring 2017. Gusty adopted many attributes from Karen's class. Karen's organization and material has been modified as needed to fit Gusty's teaching style.

Gusty Cooper

ecooper@umw.edu

Syllabus

CPSC 240 Object-Oriented Analysis & Design Fall 2017

Instructor: Gusty Cooper
When Teaching: Trinkle B6, TR 11:00-12:50, 2:00-3:50
Office and Hours: Trinkle B32, TR 8:00-10:50, 1:00-1:50, 4:00-4:50 or by appointment
Email: ecooper@umw.edu

Course Description

This course takes you to the next level as a programmer. You mastered Java basics in CPSC 220. You know how to write a class, create methods, define variables, construct algorithms, and you have worked on some challenging assignments, but perhaps it felt contrived and textbook-oriented. In this class, we focus on skills required to be a professional software developer, using Java – one of the most popular programming languages on the planet. We learn the Java API, object discovery, using UML to create OO designs, JavaFX for GUIs, design patterns, unit testing, version control, and how to develop software as part of a team. Get ready for an exciting semester as you take your programming to a more professional level!

Catalog Description

Theory and practice of the object-oriented software development paradigm. Focus is on major design principles such as abstraction, encapsulation, inheritance, polymorphism, aggregation, and visibility. Modeling notations for capturing and critiquing designs. Introduction to the concept of design patterns, and coverage of a catalog of common patterns. Students work in team projects to develop collaborative software solutions in an object-oriented language.

Prerequisite: CPSC 220 (grade of C or better).
Co-enrollment in CPSC 225 is not required but recommended.

Course Goals and Objectives

- Understand object-oriented analysis, design, implementation, test, debug, and documentation on original large-scale programs.
- Understand design techniques for identifying classes and methods.
- Understand UML class diagrams, sequence diagrams, and collaboration diagrams
- Understand encapsulation, visibility, inheritance and polymorphism in object oriented design
- Understand software development in a collaborative, team environment.
- Understand design patterns in software development.

Across the Curriculum Goals

This course counts toward the Writing Intensive Requirement, which has the following learning goals.

- (Ideas): Students will demonstrate satisfactory knowledge of the varying strategies to convey arguments, main ideas and support/evidence.

- (Organization): Students will demonstrate satisfactory knowledge of the varying patterns of composition organization and development.
- (Rhetorical Situation): Students will demonstrate satisfactory knowledge of the audience, the role of the writer, and rhetorical strategies.
- (Editing): Students will demonstrate satisfactory knowledge of writing conventions and corrections.

Required Materials
This course uses the following required material.
- Object-Oriented Design & Patterns ISBN: 9780471744870
- Course Pack

Course information, handouts, and assignments are available on Canvas. The "Pages" area of groups files by course modules and allows you to quickly locate items relevant to the current week of classes.

Class Participation and Responsibilities
You are expected to attend class. You are responsible for the content of reading assignments, lectures and handouts, as well as announcements and schedule changes made in class. If you must miss a class, check with your instructor or another student to get what you missed.

Attendance is important in this course because there are days when teams have in-class time to work together. If you are not there, it affects you and your teammates. Please be respectful of your classmates and try to attend every class period.

There is no class participation grade, but as part of the group project, students evaluate their teammates' participation and contribution to the project. These evaluations form 30% of your group project grade.

You are responsible for checking your email **DAILY** for updates or modifications to assignments, lectures, etc. I use Canvas to send announcements. It is your responsibility to edit your profile in Canvas so that messages are sent to your preferred contact method.

Assignments
All assignments are posted on Canvas. The Canvas "Syllabus" link provides a list of assignments sorted by due date.

Individual Programming Projects
Individual programming projects allow you to practice concepts discussed in class. There are two individual programming projects. Take advantage of the opportunity to practice – use as many in-class concepts as possible. Ensure you understand all programming techniques in your code because I often ask you to explain portions. You can use programming techniques found on the web, but you must cite the source of the information/code responsibly.

Late programs as well as programs that don't compile will receive a grade of zero. To receive a grade of 85% or higher, your program must include all required functionality and work correctly.

To receive a grade higher than 85%, your program must also be well designed and well documented. After all, our course is titled *Object-Oriented Analysis and Design*.

Group Project

Most software development professionals spend their time working in groups. They design software in groups, they divvy tasks among group members, and develop the pieces in tandem. Working with others can be challenging, but it is a skill you should practice. This course is one of your first opportunities to practice group work skills.

The group project typically receives a group grade; however, if you do not contribute, your grade will be lower. Part of the group project includes evaluating yourself and your teammates confidentially. The evaluations factor into your individual project grade.

If teammate comments and my observation indicate that you have not actively contributed to your group's code, you cannot receive a passing grade on the group project.

Labs

We have about 15 labs this semester. We typically start a lab in-class on Thursday and it is due the following Tuesday. All due dates are posted on Canvas. I recommend using your laptop for in-class labs because it allows you to continue working easily after the in-class portion. You may use the computers in B12.

Labs include combinations of Java programming practice, use of design tools, self-exploration, and questions. On some labs, I actively lead an exercise, where you can follow along.

You are expected to attend the lab period unless you have already completed and submitted your assignment to Canvas.

Labs are graded as complete/incomplete based on demonstrated effort.

Writing Assignments

CPSC 240 is a writing intensive course and earns you credits toward the writing-across-the-curriculum graduation requirement. In this course, you will receive significant practice writing in the computer science discipline. There will be three formal writing assignments during the semester. You will receive instruction in the editing process and have the opportunity for revision of submitted work.

Students may obtain help with the organization and/or the writing mechanics of any writing assignment from the UMW Writing Center. You can visit the Writing Center multiple times for each paper. The Writing Center staff is incredibly friendly and helpful. You can also schedule appointments with me to discuss writing assignments.

Weekly Quizzes

A short 15-minute quiz will be given most Thursdays. The quiz will cover the course material covered the previous week during class. The lowest quiz grade will be dropped when determining final course grades.

NO makeup quizzes will be given except in the case of an unavoidable absence that can be verified as legitimate through the Office of Academic Services.

Final Exam

The cumulative final exam is scheduled for XXX.

In the case of a planned absence, alternate exam arrangements must be made in advance by notifying the instructor.

The Honor Code and CPSC 240

All assignments in this course fall under the conditions of the Honor Code. The CPSC department honor code can be found here and is linked to the Canvas course page:

http://cs.umw.edu/mediawiki/index.php/CPSC_Department_Honor_Code _Policy

The information below supplements the department's honor code policy and clarifies how the honor code applies to assignments in this section of CPSC 240.

- Writing a computer program is a creative endeavor like writing a paper. Plagiarizing code in a program is an honor code violation just like plagiarizing words in a paper.
- For all submissions, be prepared to explain how the program works or how you arrived at your answer if asked to do so.
- All CPSC programs submitted for class assignments may be subjected to automated plagiarism tools such as Moss.

If you have any questions or concerns about how the Honor Code applies to this course, please contact me at any time.

The Honor Code & Weekly Labs

Labs are designed for learning. You can freely consult with others – ask questions, discuss, and share ideas. The others can be me, CPSC 240 students, CPSC lab aids, the Internet, other CPSC students, and other CPSC professors. You must write or type solutions independently. You cannot share code or portions of code electronically.

The Honor Code & Programming Projects

The two individual programming assignments must be completed individually. Do not ask others for assistance with your individual assignments or work with others on these assignments.

- You cannot copy code from another student. Copying includes copying the idea while modifying such things as variable names, method names, spacing, etc.
- You cannot provide your code to another student for the purposes of copying either explicitly or by copying and modifying such things as variable names, method names, spacing, etc.
- You cannot copy code from any Internet site and include it in your program without citing the source of the code in a comment within your program.
  Example:
  ```
  // bubble sort algorithm on lines 65-85 was copied from
  // http://mathbits.com/MathBits/Java/arrays/Bubble.htm
  ```

In this course, you do not have to cite the Java API when using classes/methods/etc. from the standard library.

Group programming assignment requires working with teammates, which includes collaborative ideas and shared code. Teams can share within, but must not share with other teams.

All computer-programming assignments shall be submitted with an accompanying UMW honor code pledge. This pledge acknowledges that you, alone, have created the code. By submitting files online, it is implied that you have submitted the assignment with the honor code pledge.

The Honor Code & Exams and Quizzes

Exams and quizzes must be completed individually, without consulting other students, written sources (notes and books), and electronic sources (your phone, the Internet, etc.).

All exams and quizzes must be submitted with a handwritten, signed UMW honor code pledge.

This pledge acknowledges that you have not given or received assistance while completing the assignment.

Disabilities

The Office of Disability Resources has been designated by the University of Mary Washington as the primary office to guide, counsel, and assist students with disabilities. If you have a disability that may affect your participation in this course, please contact your instructor as soon as possible to discuss your approved accommodations. I will hold any information you share with me in strictest confidence unless you give me permission to do otherwise.
If you have not contacted the Office of Disability Resources and have accommodation needs, I will be happy to help you contact them. The office will require appropriate documentation of a disability.

Grade Determination

Final Grades

Final grades are based upon the following.

- Group Project – 20%
- Individual Programming Projects – 14%
- Labs – 12%
- Writing Assignments – 14%
- Quizzes – 20%
- Final Exam – 20%

Passing Grade Criteria

To receive a passing grade in this course, students must satisfy **all** the following:

- Earn an average of 60 or better on quizzes and final exam.
- Successfully complete all programming projects and earn a programming average of 60 or better.

- Successfully complete all formal writing assignments and earn a writing assignment average of 60 or better.

Final letter grades will be determined according to the following scale.

A 93-100%	B+ 87-89%	C+ 77-79%	D+ 67-69%
A- 90-92%	B 83-86%	C 73-76%	D 60-66%
	B- 80-82%	C- 70-72%	F below 60%

Mid-semester Grades

The University provides the opportunity to provide grading feedback midway through the semester. A student will receive a mid-semester unsatisfactory (U) grade if he/she has a 65 average on writing assignments or programming assignments or quizzes. Any student receiving a U mid-semester grade should meet with me to develop a performance improvement plan.

Course Agreement – Completed, Signed, and Returned by Each Student

My signature below acknowledges that I have read and understood the material in the syllabus for CPSC 240, spring 2017. I also acknowledge the following.

- I have completed the course prerequisite CPSC 220 with a grade of C or higher.
- I understand that to be pass CPSC 240,
    - I must have a passing average (60% or better) on quizzes and the final exam.
    - I must have a passing average (60% or better) in the programming assignments.
    - I must have a passing average (60% or better) in the writing assignments.
    - I must complete the group project assignment.
- I am expected to attend class during every scheduled class meeting. I am responsible for all assigned readings and all materials presented in class lectures. I am responsible for obtaining all materials and notes for any classes for which I am absent or arrive late.
- I am aware of Gusty's office hours. I understand that if I need Gusty's help but cannot attend during office hours, I may send an email to establish an appointment at a mutually agreeable time. I understand that I can email questions to Gusty, and he will do his best to respond in a timely fashion, noting that some questions require conversation to answer. If I need immediate assistance, I may consult a lab aide or another CPSC faculty member.
- I have read and understand the honor code and information about how it applies to all assignments in this course.
- I understand that to achieve the highest level of success in this course, I should regularly attend class, actively listen and take notes during lecture, ask for assistance in a timely manner when topics are unclear to me, begin all assignments on the day they are assigned, and review any errors that I make on assignments after the graded assignment has been returned.
- Gusty will provide class presentations and materials pertinent to the course. Gusty will distribute all assignments in a manner that permits adequate time for students to complete them. To help students, Gusty will be available during posted office hours (unless otherwise announced) and reply to emails. Gusty will grade and return assignments in a timely fashion.

Print Student Name: ______________________________Date: _______________

Student Signature: ____

Course Schedule – Assignments and Due Dates

• Module 1 – Java Review **C1,29** • Syllabus • CPSC 240 vs. CPSC 220 • Writing 1 Assigned • C1JavaBasics.docx • Read Chapter 1 of Horstmann	• Module 1 – Java Review **C2,31** • C2JavaDataTypes.docx • Lab1JavaAPI.docx
• Module 1 – Java Review **C3,05** • Lab 1 due • Project 1 Assigned • C3JavaDoc.docx • Read Chapter 2 of Horstmann	• Module 2 – OOD Processes&Tools **C4,07** • Quiz 1 • C4OOD&ObjectDiscovery.docx • Lab2ClassesJavaDoc.docx
• Module 2 – OOD Processes&Tools **C5,12** • Project 1 Plan due • Lab 2 due • C5UseCases&CRCCards.docx • Read Chapter 3 of Horstmann	• Module 2 – OOD Processes&Tools **C6,14** • Quiz 2 • Writing 1 due • Writing 2 Assigned • Lab3UseCases.docx
• Module 2 – OOD Processes&Tools **C7,19** • Lab 3 due • Book selection due • Discuss Project 1 • C7ClassDiagrams.docx	• Module 2 – OOD Processes&Tools **C8,21** • Quiz 3 • Module 2 – C8SOLID.docx • Lab4ClassDiagrams.docx
• Module 2 – OOD Processes&Tools **C9,26** • Lab 4 Due • Project 1 due – in-class demonstration • Project 2 Assigned • Lab5DesignDecisionTradeoffs.docx	• Module 3 – Java's OO Features **C10,28** • Quiz 4 • Lab 5 Due • Module 3 – C10Inheritance.docx • Lab6Inheritance.docx

• Module 3 – Java's OO Features **C11,03** • Lab 6 due • Project 2 Plan due • C11JavaFX.docx • Lab7JavaFX.docx • Read Chapter 4 of Horstmann	• Module 3 – Java's OO Features **C12,05** • Quiz 5 • C12Interfaces&Polymorphism.docx • Lab8Interfaces&Polymorphism.docx
• Module 3 – Java's OO Features **C13,10** • Lab 8 due • Writing 2 In-class Discussion • Lab9GroupQuestionnaire.docx • C13UnitTesting.docx	• Module 3 – Java's OO Features **C14,12** • Quiz 6 • Project 2 due – in-class demonstration • Writings 2 Discussion Reflections due (on F) • Lab10UnitTesting.docx
• No class, Fall Break **FB,17**	• Module 3 – Java's OO Features **C15,19** • Quiz 7 • Lab 10 due • Writing 2 due • Writing 3 Assigned • C15VersionControl.docx • Lab11VersionControl.docx
• Module 4 – Design Patterns **C16,24** • Lab 11 due • Project 3 Assigned • Teams announced • C16VideosForGroup.docx • C16DesignPatterns&StrategyPattern.docx • Read Chapter 5 of Horstmann	• Module 4 – Design Patterns **C17,26** • No quiz • In-class reflective activity • C17ObserverPatternMVC.docx • Lab12ObserverPattern.docx
• Module 4 – Design Patterns **C18,31** • Lab 12 due • C18AbstractClasses&TemplatePattern • Read Chapter 6 of Horstmann	• Module 4 – Design Patterns **C19,02** • Quiz 8 • Writing 3 due • C19DecoratorPattern.docx • Lab13TemplatePattern.docx

• Module 4 – Design Patterns **C20,07** • Lab13 due • C20IteratorPattern.docx • C20FactoryPattern.docx	• Module 5 – Parallelism **C21,09** • Quiz 9 • C21to23Parallelism.docx • Lab14IteratorPattern.docx • Group work time
• Module 5 – Parallelism **C22,14** • Lab 14 due • C21to23Parallelism.docx • Group work time	• Module 5 – Parallelism **C23,16** • Quiz 10 • Lab15Parallelism.docx • C21to23Parallelism.docx • Group work time
• Module 5 – Parallelism **C24,21** • Lab 15 due • Group Progress Check due • C24to26Concurrency.docx • Group work time	• No class, Thanksgiving Break **TGB,23**
• Module 5 – Parallelism **C25,28** • C24to26Concurrency.docx • Group work time	• Module 5 – Parallelism **C26,30** • No Quiz • C24to26Concurrency.docx • Group work time
• Project 3 due **C27,05** • Group project demonstrations	• Group project demonstrations **C28,07**

Class Introduction

The Course Syllabus describes our class.

Canvas Organization

Canvas is our learning management system for course material and assignments. CSPC 240 Sections 2 and 3 both view the same Canvas material.

Home

Takes you to the home page of our Canvas course

Announcements

Contains announcements that I post about our course

Syllabus

Contains the Canvas version of our Syllabus. The Canvas Syllabus provides a chronological listing of assignments. The Canvas Syllabus also contains a link to the syllabus distributed on the first day of class

Assignments

Contains our class assignments.

Pages

Contains the Modules of CPSC 240. There are five modules. Pages has five links – one for each module. Each link is a Canvas page, which has narrative along with links to external references and Canvas files. The course pack that contains the same information as the Canvas page.

- Module 1: Java Review
- Module 2: OOD Processes and Tools
- Module 3: Java's OO Features – inheritance, interfaces, abstract classes, polymorphism
- Module 4: Design Patterns
- Module 5: Parallelism and Multithreading
- Version Control & Tools
- Group Work

Discussions

Contains class discussions. Some assignments may have questions and comments that can be discussed using Canvas' discussions.

Files

The main folder on Canvas contains subfolders with the course information.

- CoursePack.pdf – a course pack with all material collected into a book-like structure.
- Module1JavaReview – contains Module 1 lecture notes.
- Module2OODProcessesTools – contains Module 2 lecture notes.
- Module3JavasOOFeatures – contains Module 3 lecture notes.
- Module4DesignPatterns – contains Module 4 lecture notes.
- Module5Parallelism – contains Module 5 lecture notes.
- Labs – contains labs.
- Projects – contains projects.

- Writings – contains writing assignments.
- Code – contains sample code.

Lecture Notes

Canvas contains individual files (labs, programming projects, writing assignments, lecture notes) that have been collected into our course pack, which can serve as a printed reference. Lecture notes contain most of the important information discussed in class. The idea is that you can follow the class lecture with the lecture notes, filling in blanks with pertinent terms and phrases. This approach saves you from having to frantically take notes during lectures. Printed lecture notes are helpful for following class lectures – you can fill in the blanks with a pencil. As we discover errors in the course material, I will update the files on Canvas.

CSPC 220 and CPSC 240

I think of computer programming as problem solving. CPSC 220 and 240 are designed to create your fundamental problem solving skills in Java; however, the skills can be transferred to most programming languages. CPSC 220 focuses on lower-level, algorithm-creating, problem solving skills and CPSC 240 focuses on higher-level, organizational, problem solving skills.

The focus of CPSC 220 is solving problems by creating algorithms within Java classes. For example, you learn to solve a problem such as create a method with an array parameter that returns the largest element in the array. In CPSC 220 you learn how to create algorithms using input/output, conditionals, while loops, for loops, for-each (enhanced for) loops, methods, parameters, etc. You learn how to create introductory Java classes such as a Person with fields (e.g., name, age) and methods (e.g., getName, canVote). You also learn problem solving techniques such as elementary sorting algorithms. You can think of CPSC 220's focus as down in the weeds of programming.

CPSC 240 builds upon these low-level, algorithmic problem solving skills, but the focus of CPSC 240 is discovering classes and their relationships within a problem. You still need your weed-level skills and they will improve in CPSC 240, but we want to learn how to read a large problem, understand the problem, and design an organizational structure for the problem. We study tools (UML) and patterns (Strategy, Iterator, Observer) that help our discovery process. In CPSC 240, students are mature enough to discover information on their own.

Module 1 – Java Language Review

Module 1 Overview

In module 1, we spend several classes reviewing Java and programming concepts that you have studied previously. This review allows everyone to re-condition their programming skills and get ready to expand them. The following subsections provide links that you can use for re-conditioning your programming skills.

Oracle Java Tutorials

Even though you have Java programming experience, the Oracle Java Tutorials contain illuminating information. For example, the Lesson on Object-oriented Programming discusses the basics of OOP as conducted in Java – Objects, Classes, Inheritance, Interfaces, and Packages.

https://docs.oracle.com/javase/tutorial/

If your Java programming needs refreshing, you should start with the trail *Learning the Java Language* (https://docs.oracle.com/javase/tutorial/java/index.html) and paying special attention to sub-trail *Java Basics*, which covers variables, arithmetic, Booleans, ifs, loops, etc. (https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html).

Useful references

The Java API:
http://docs.oracle.com/javase/8/docs/api/

Java essentials syntax/usage sheet:
http://introcs.cs.princeton.edu/java/11cheatsheet/

How to write JavaDoc comments:
http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

videos about JavaDocs in intelliJ:
https://www.youtube.com/watch?v=CAexSdMCuGg

Exceptions tutorial:
http://docs.oracle.com/javase/tutorial/essential/exceptions/index.html

Lectures

- C1JavaBasics.docx
- C2JavaDataTypes.docx
- C3JavaDoc.docx

Labs

- Lab1JavaAPI.docx
- Lab2ClassesJavaDoc.docx

Writing Assignment

- Writing1IDEUsersGuide.docx

Class 1 – Java Basics

Learning Objectives

- Review the software development process.
- Review Java programming with classes and objects.
- Review Java programming with a main method that uses classes and objects.

Assumptions

You have some experience using Java from a previous course.
If this is NOT true, be sure to speak with Gusty after class today!

Software Development Process

Software is developed to solve a problem. All problems must be ________________ before they can be solved. This means there is a statement (or collection of statements) describing the problem to be solves.

Java Programs

Fact: Java was designed to support object-oriented (OO) programming.

Can you write a non-OO program in Java? That depends on what you consider non-OO. If you consider a non-OO program to be one without classes, then you cannot write a non-OO program in Java Every project that you create must include at least one ____________ and that class must include at least one _______________________.

However, you can write Java projects that largely ignore the principles of OO design, and you may have done that to a certain extent in your previous course. When you create a program with a `static main` and other supporting methods, you are not creating an OO design. In this course you learn how to create a good OO design and practice applying these ideas to projects – starting with some small items and working toward larger scale projects culminating in our group project.

For larger projects, embracing OO design results in a well-organized program that is more easily understood. In OO design, your goal is to divvy your project description into a series of customized ________________________________ that represent real world objects. These customized data types are called _________________ and the variables created of these types are called __________________________.

Java Project Organization

A project is created by coordinating the behavior of data types created with classes.
Each class that is used in the project resides in its own ______________.
Each class can contain both __________ members and _____________.

The name of the file must be __________________ as the name as the class (identical spelling & capitalization).

```
public class Greeter {
      //constructor
      public Greeter (String aName) {  name = aName; }

      //method
      public String sayHello( ) {  Return “Hello, ” + name + “!”; }

      //data member
      private String name;
}
```

What’s the purpose of a **constructor**?

What’s the purpose of a **method**?

Why would a class include data?

Can you execute the class above alone? ________________! The class just defines a data type `Greeter`.

Where does execution of a Java program start?
Execution of a Java program starts with the class that contains the `main` method. To execute the method/s in the `Greeter` class, we need to create a class that contains a `main` method that will to instantiate an object (or objects) of type `Greeter` and make the calls to the methods of the Greeter class.

If we’re creating a class strictly for the purposes of testing a second class, we often refer to the class as a _________________ class. Here’s a tester class for the Greeter.

```
public class GreeterTester {
      public static void main (String[ ] args) {
            Greeter worldGreeter = new Greeter (“World”);
            String greeting = worldGreeter.sayHello( );
            System.out.println(greeting);
      }
}
```

We can include a `main` method in our `Greeter` class, but this is awkward and not part of OO design.

The signature for the main method always looks like the one above:

```
      public static void main (String[ ] args)
```

You've undoubtedly encountered this signature in projects that you've written. Because many popular IDEs include this method header automatically, you may not have paid much attention to the details.
What does the reserved word `public` indicate?
What does the reserved word `static` indicate?
What does the reserved word `void` indicate?
What does the parameter identifier `args` represent?

Test Yourself

1. Can non-object oriented programs be written in Java?
2. A class is a way of defining a customized _________ __________.
3. True or false? You can name a Java file anything that you want without regard to the contents of the file.
4. A class created for the sole purpose of testing out methods/operation of another class is called a ______________ class.
5. Consider the `main` method definition - `public static void main (String[] args)`
   What does the word static indicate about the method?

   What is `String[] args`?

class, method, data types, classes, objects, file, data (or field), methods, the same, no, tester, no/yes, false, tester

Class 2 – CPSC 240 Java Data Types

Learning Objectives Readings

- Understand that Java objects are implicit pointers
- Understand the parameter passing possibilities in Java

Primitive Types, Reference Types, and Objects

Java allows programmers to declare variables of two classifications: *primitive types* (which do not have associated objects) and *reference types* (which have associated objects).

Primitive types are the types "built in" to the Java language.
`int, double, char, boolean, float, ..`

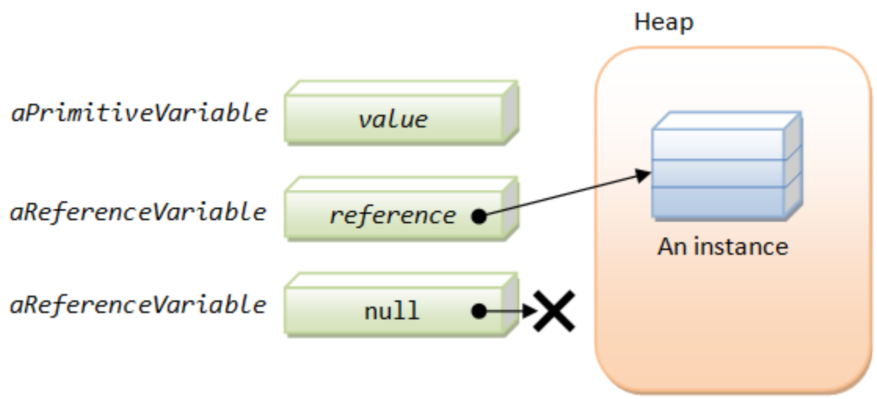
Reference types are defined in classes. Classes allow us to define more complex items.

- Classes can be ___________ (live in your own personal folder) or can be placed in a library and shared with other programmers (for example, in the ________________________).

- Variables whose type is defined by a class reference **objects** of the class.

- When you declare an object, you are __________________ the class.

You have probably used primitive and reference types in past projects, but you may not have thought about how they are handled differently in the language.   Let's think about that now.



Rules related to declaring variables using primitive types:
Example: `int sum;`

- All primitive types start with a lower-case letter (`int, boolean`…)
- Primitive types are NOT implemented with classes.
- Variables declared with a primitive type do not have to be instantiated.
- A fixed amount of memory is set aside for each variable whose type is a primitive, so there is a maximum/minimum value for each of these types (the maximum/minimum value that can be stored in that amount of memory space).

Example: `int sum;`
What does the memory allocation look like after this declaration?

Question: How could you find out the maximum or minimum value that a primitive numeric Java data type?

Rules related to declaring objects (variables created using a class):
- All classes included in the Java API start with a ____________________ (`Integer, ArrayList, ….).` Programmers also usually start their class names with a capital letter for consistency.

Question: Will your code compile if you write a class using a lower-case letter for the class name?   Example: `class greeter { …. }`

- When an object is declared, the computer sets aside space to store the address of the (yet to be determined) location of the actual object.

  Example: `ArrayList  myList;`
  What does the memory space associated with this variable look like?

- The space for an object is created when a class type is instantiated, which involves using the “new” operation.
  Example: `Integer myNumber = new Integer();`

- When the “new” operation executes, the space for the object is allocated and the address of that reserved space is associated with the object’s identifier.

- When the “new” operation executes, the constructor for the class is called. If no arguments are provided, the ______________constructor is called.
  Example: `Integer myNumber = new Integer();`

- If one or more arguments are provided, the version of the constructor that requires that number of parameters is called.
  Example: `ArrayList  myList = new ArrayList(10);`

- Thus, when objects are used in Java, a special type of variable known as a ___________________ is created. This means that the value associated with the identifier is a memory address. It’s the memory address of the actual location of the data.

Object References

Suppose that you declare an object in Java. The information associated with the identifier is a reference to an object (the address of the object) not the object itself.

In Java you’re always using pointers to objects, but this fact is hidden from you. These are known as *implicit pointers.*

Implicit pointers are nice because they hide a lot of memory management tasks from the programmer, BUT they can also lead to odd errors if the programmer isn’t cautious.

EX: Suppose we have a Greeter class[1]

```
public class Greeter {
   private String name;

   /**
    * Constructs a Greeter object for greeting people
    * @param aName name of person to be greeted
    */
   public Greeter(String aName) {
      name = aName;
   }
```

[1] Greeter code from Horstman book.

```
   /**
      Greet with a "Hello" message.
      @return "Hello" and name of greeted person.
   */
   public String sayHello() {
      return "Hello, " + name + "!";
   }
}
```

And we create a main program that declares objects of type Greeter:

```
public class GreeterTester {
   public static void main(String[] args) {
      Greeter worldGreeter = new Greeter("World");
      Greeter anotherGreeter = worldGreeter;
      anotherGreeter.setName(“Karen”);
      String greeting = worldGreeter.sayHello();
      System.out.println(greeting);
      System.out.println(anotherGreeter.sayHello();

   }
}
```

What is printed?

Why?
What if you want to make a copy? Use the copy constructor, which is implemented as the clone method (more about this later).

Tips:

Avoid using the = operator with objects because

Avoid using the == operator with objects because

Passing Parameters

The object that calls a method is called an *implicit parameter*. Sometimes the implicit parameter is referred to as the *calling object.*

The parameters in the parameter list (if any) are *explicit parameters*.

The values passed into a method are called *arguments*.

EX:
```
anotherGreeter.setName("Dave");
```

What is the implicit parameter?

What is the argument?

What is the explicit parameter?

If you need to refer to the implicit parameter (calling object), use the reserved word **this**. It's a reference to the calling object.

```
class Greeter {
     ...
      public void setName (String name) {
            this.name = name;
      }

            //data member
          private String name;
     }
```

Review

1. This line of code instantiates an object of the Greeter class:
   `aGreeterObject.setName("Joe");`
   a. True or false?

2. Objects are reference variables in Java.

3. Write a call to the default constructor for the ArrayList class:

4. You should be cautious when you use the assignment and equality operators on objects in Java. True or False?

5. Which of the following are shown in this line of code: implicit parameter, explicit parameter, argument?
   `anObject.doSomethingAwesome(4, "Joe");`

local, Java API, instantiating, Integer.MAX_VALUE (or Google), capital letters, yes, default, reference (or pointer), false, true, true

Class 3 – JavaDoc Documentation

Learning Objectives

- Understand the advantages of using Javadoc comments in Java
- Understand common Javadoc tags including `@param`, `@return`, and `@author`
- Understand when to use regular comments instead of Javadoc comments

JavaDoc Introduction

Java introduced a tool to produce HTML pages of documentation for your programs. This is super convenient because it lets you share the essential details of your program in a standard, easy to read way -and it doesn't take a lot of extra effort.

All you need to do is use "_______________" style comments in your program. Then use the Javadoc tool to generate the .html pages automatically from your Java source files.

Javadoc was used to document all standard Java packages in the ____________________ (Application Programming Interface). All are available on the web. Because developers are used to reading the API, they will find your documentation easy to __________________and __________________ because you'll be using the same format.

Writing Javadoc comments for your own classes allows you to

- ____________________ when creating documentation.
- easily _________________________ because the documentation is stored in the code itself.
- maintain documentation using the ____________________________________. This is easy for other developers to understand.
- __________________________ about your created data types via the web.

Rules:

- Javadoc comments are delimited by `/** .... */`
- Javadoc comments always _______________ the code that they describe.
- The first sentence of each Javadoc comment will be copied into your Javadoc .html files. You need to start the sentence with a __________________ and end it with a ________________________.

- You should provide a general description of your class as a ___________________________.

  Example:

```
/**
 * Class stores a name and creates a customized greeting.
 *
 * Additional descriptive text is included here.
 *
 * @author Kay Horstman
 * @version 1.0 6/6/2017
*/
```

```
public class Greeter {
```

- You should provide a separate description of each of your methods. For each method, you should also comment information about _______________ and ____________________________. To do this, use the `@param` and `@return` tags.
  Example:

```
   /**
    * Greet with a "Hello" message.
    * @return "Hello" and name of greeted person.
    */
   public String sayHello() {
      return "Hello, " + name + "!";
   }
```

Example:

```
   /**
    * Constructs a Greeter object for greeting people
    * @param aName name of person to be greeted
    */
   public Greeter(String aName) {
      name = aName;
   }
```

JavaDoc Standards

Java standards say that every class, every method, every parameter, and every return value should have a comment.

JavaDoc Tags

Some commonly used Javadoc tags are:

```
@param [parameter name] [parameter description]
@return [return argument description]
@throws [exception thrown name]
@exception [exception thrown] [exception description]

@since [version]
@deprecated [description]
@see [hyperlink]
```

In each of these tags, don't type the square brackets and instead fill in the indicated information.

Generating JavaDoc HTML

- To generate the Javadoc in Unix, type `javadoc *.java`
- The Javadoc tool will produce a .html file for each of your .java files, an index.html file, and a few summary files.

- Most IDEs have a menu option that will generate the Javadoc for you. Google to find out how to do this in your preferred IDE.
- Gusty will demo the process during class using IntelliJ (the free community edition)
    - `Tools > Generate JavaDoc…`

Note: For CPSC 240 you are not required to host the JavaDoc for your project publically. You can if you like and might find the cs.umw.edu server to be a good hosting location, or you might like to sign up for a Domain of One's Own, or host using a different hosting service.

Pass by Value and Pass by Reference

All explicit parameters are passed by value in Java. This means that a copy of the value of the parameter is passed in to the method. If the parameter is a primitive, any changes to the parameter's value are not "saved" by the method.

EX:

```
void update (int sum, int value) {
      sum = sum + value;
}
```

However, if the parameter is an object, a copy of the object's memory address will be passed into the method. When the object's identifier is referenced in the method, it points back to the original memory location. Thus, any changes to the object's referenced value will be saved when the method completes.

```
void update (ArrayList myList, int value) {
      myList.add(value);
}
```

JavaDoc, Java API, read, understand, program, create, JavaDoc comments, tell others, precede, verb, period, parameters, return value,

Module 2 – OOD Processes and Tools

Module 2 Overview

In this module, we study processes and tools for creating an OO design. We study the following techniques.

- Noun-Verb walk-through, which is an intuitive way to identify objects and methods in a software specification.
- SOLID – this is an acronym that stands for the following.
    - S – Single responsibility – a class should have a single responsibility
    - O – Open-closed principle – objects should be open for extension, but closed for modification.
    - L – Liskov substitution principle. If S is a subtype of T, the objects of type T in a program may be replaced with objects of type S without changing the correctness of that program.
    - I – Interface segregation principle – A class using an interface is a client of the interface. A client should not be forced to depend on methods it does not use. If an interface has lots of methods and a client uses a few, the client may have to be updated when the interface changes methods the client does not use. To avoid this, interfaces are split into smaller interfaces.
    - D – Dependency inversion principle – depend on abstractions, not concretions.

  We will study portions of SOLID.
- Unified Modeling Language (UML), which allows you to create a visual representation of a software system. UML has many drawings, and we will study several.
    - Use Case Diagram
    - Class Diagram
    - Sequence Diagram

References

The following subsections provide links to various supplemental material for this module.

General OO

Basics of OO terminology: https://docs.oracle.com/javase/tutorial/java/javaOO/

Why is it good to split a program into multiple classes:
http://programmers.stackexchange.com/questions/154228/why-is-it-good-to-split-a-program-into-multiple-classes

Video on Object Oriented Design (requires login to UMW library to view):
http://umw.kanopystreaming.com.ezproxy.umw.edu/video/classes-and-object-oriented-programming
(Note this is in Python, but the ideas all extend to Java)

Tutorial on OOD/OOP:
http://atomicobject.com/pages/OO+Programming

Overriding the equals method: https://www.sitepoint.com/implement-javas-equals-method-correctly/

SOLID

SOLID tutorial: http://code.tutsplus.com/series/the-solid-principles--cms-634
Solid http://www.kirkk.com/modularity/2009/12/solid-principles-of-class-design/
SOLID: http://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/
SOLID: https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
SOLID: http://howtodoinjava.com/2013/06/07/5-class-design-principles-solid-in-java/

edX UML Course
https://www.edx.org/course/uml-class-diagrams-software-engineering-kuleuvenx-umlx

UML Use Case

Tutorials

http://www.visual-paradigm.com/tutorials/writingeffectiveusecase.jsp

https://www.youtube.com/playlist?list=PL05DE2D2EDDEA8D68

http://csis.pace.edu/~marchese/CS389/L9/Use%20Case%20Diagrams.pdf

Videos

https://www.youtube.com/watch?v=tLJXJLfLCCM

https://www.youtube.com/watch?v=_JCsqdNf8bA

UML Class Diagram

Tutorials

http://www.objectmentor.com/resources/articles/umlClassDiagrams.pdf

Discussion on Types of Class Relationships

http://nirajrules.wordpress.com/2011/07/15/association-vs-dependency-vs-aggregation-vs-composition/

Software Tools for Creating UML

You need a tool to draw UML diagrams. You could use PowerPoint or Google Draw, but this would force you to create the various UML shapes. You want a tool that easily draws the shapes.

You can use any tool that you'd like to create the diagrams; however, I recommend https://www.draw.io. Draw.io is a web-based tool that you can use for free. Draw.io supports UML and other drawing symbol packages. Draw.io allows you to save your drawing a .xml file on your computer, which can be reopened in Draw.io. Draw.io also allows you to save your drawing as a PDF file on your computer, which can be viewed, printed, and submitted on Canvas.

We have Microsoft Visio and Rational Rose installed in B12. Both tools are commonly used in industry and would be nice skills to add to your resume. However, they are more complex than Draw.io, and I am not familiar with them.

Gliffy is another web-based tool – http://www.gliffy.com/uses/uml-software

Creately is another web-based tool – https://creately.com/

Lectures

- C4OOD&ObjectDiscovery.docx
- C5UseCases&CRCCards.docx
- C6ClassDiagrams.docx
- C7SOLID.docx

Labs

- Lab3UseCases.docx
- Lab4ClassDiagrams.docx
- Lab5DesignDecisions&Tradeoffs.docx

Writing Assignment

- Writing2BookReport.docx

Class 4 – OO Design and Object Discovery

Object-Oriented Design

Learning Objectives

- Understand the difference between a class and an object.
- Understand how to use the following terms related to object-oriented design: fields or data members, methods, encapsulation, state, behavior, and identity.

Object-Oriented Design

Object Oriented Design is a method of

Class is code, ADT is idea

Class = the implementation of an ADT ____________________

= data and methods, encapsulated _____________together

Object = instantiation of a class
= has a
state (___________________________________),

behavior (_________________________________),
and identity (a variable name)

The methods can be used to change the state of the object over time.

Object Discovery

Learning objectives:

- Understand how to use standard strategies to identify possible classes, responsibilities, and relationships in each project description.
- Understand how to use Use Cases and CRC cards in the early design process.

Identifying Classes

Rule of thumb for identifying classes in a project description
Look for the nouns. Not every noun will be a class, but some nouns will be classes.

UMW Room Reservation System Problem Description

UMW is in desperate need of scheduling software that would allow administrators, faculty, and student groups to reserve rooms for class meetings, meetings, and other functions on campus. The UMW community should be able to view a calendar that shows reserved rooms on any day. The calendar should note the times that each room is reserved and person who requested the reservation. Users should be able to search for available room based on room capacity (i.e. I need a room that will hold 30 or more people), building (i.e. I need a room in Trinkle Hall), date/time (i.e. I need a room on Tues Feb 8 from 3:00-5:00), AV equipment (i.e. I need a room with a projector), and seating (i.e. I need a room with tables not desks). The system should return a list of available room options and the user should be allowed to reserve a room from this list.

Write a list of possible classes that might be contained in the final system:

```
/**
 * Stores information about a user of a software system
 * @author CPSC 240 Instructor
 */
public class LoginAccount {

    Person p;
    String userName;
    String passWord;

    /**
     * Constructor initializes fields
     * @param person person with login account
     * @param userName user name
     * @param passWord pass word
     */
    public LoginAccount(Person p, String userName, String passWord) {
        this.p = p;
        this.userName = userName;
        this.passWord = passWord;
    }

    public String getUserName() { return userName; }

    public String getPassWord() { return passWord; }

    @Override
    public String toString() {
        return "LoginAccount{ " + userName + " " + passWord + " };

    @Override
    public boolean equals(Object o) {
        if (o == null || getClass() != o.getClass())
            return false;
        else if (this == o)
            return true;
        else {
            LoginAccount la = (LoginAccount)o;
            return (userName.equals(la.userName) &&
                    passWord.equals(la.passWord);
        }
    }
}
```

Unified Modeling Language (UML)

UML is a tool with various shapes and arrows that is often hard to remember. If you used UML daily these shapes and arrows would become more intuitive. UML has two purposes.

1. Discovery – use UML to __________ the design solution of a problem. When discovering a design solution, the pretty, detailed diagrams are not necessary. Often, you can create your own short-hand version of pseudo UML on white boards and pieces of paper. One UML tool that we studied, the CRC cards, recommend small hand written index cards for this discovery process, which you eventually discard. Many of the artifacts creating during the discovery process are discarded.
2. Documentation – use UML to ____________ the design solution of a problem. One thing you will learn is the code has a long shelf life. A good example is LINUX, a popular operating system, which began in 1992. Many programming jobs fix bugs and add new capabilities to an existing, legacy code base. New programmers must

understand the existing code base to change it. UML can provide an overview, allowing programmers to understand the design solution. In this case, a pretty diagram can be helpful. I have always thought there are three things needed for maintenance programming.

a. A requirements specification. This is the written description of the system to be created, often captured in a requirements database tool.
b. High-level diagrams of the design solution. UML is a good tool for capturing these.
c. Code.

Abstract Data Type, grouped, value of fields, methods, discover, document

Class 5 – Use Cases and CRC Cards

Use Cases

Learning Objectives

- Understand that Use Case write ups are a way of documenting functional requirements and developing an understanding of a software project.
- Gain experience in documenting Use Cases using an example format.

Understanding System Use

Part of designing a system involves developing an understanding of how the system will be used.

____________________ requirements specify ways that end users will interact with a system. During the early phases of a software project, functional requirements are often documented using a UML tool called ____________________. A Use Case is a formal way of describing an activity that your software must handle.

Documenting use cases allows analysts/software engineers to think through a project in more detail and can help them to identify fundamental components that must be present in the final software solution.

There is no standard template for writing use cases. Each company uses its own standard. However, all use cases are capture the flow of a user interacting with the system.

Use Case Format Example:

Title: (has a verb)
Short description:
Main Flow:

1. sequence of steps in “standard” operation
2. more steps

Alternate Flow A:

1. Alternate sequence of steps for such things as user error.
2. More steps

Alternate Flow B: You may have several alternate flow

The main flow and alternate flow emphasize the role of the actor (usually an end user of the system) and the system itself.

Use Case Example – Password Update

Title: Change Password
Main Flow:

1. User indicates they want to change their password
2. System prompts for current password
3. User types in current password
4. System verifies password
5. System prompts for new password
6. User types in new password
7. System checks new password is acceptable (length, special chars, etc.)
8. System prompts user to retype password
9. User retypes new password
10. System confirms new passwords match

Alternate Flow A:

3. User types incorrect current password
4. System prompts password is incorrect and to try again. Use flow returns to standard step 3

Alternate Flow B:

7. System determines User has entered unacceptable password.
8. System prompts password is weak and to enter a stronger password. User flow returns to standard step 6

Alternate Flow C:

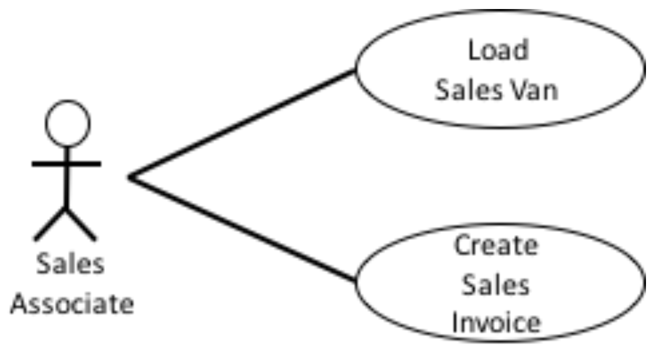
10. System determines two passwords do not match.
11. System prompts passwords do not match and to reenter. User flow returns to standard step 6.

Use Case Diagram

- Visual index to all use case write-ups for a system.
- Associate actors with actions, where an action is the use case title or a shortened version of it.

Use Case Diagram Example

The following is a single use case diagram that shows a Sales Associate has two actions – load sales van and create sales invoice.



Practice Writing Use Cases and Use Case Diagrams

Write two use cases for the room reservation system.

UMW Room Reservation System Problem Description

UMW is in desperate need of scheduling software that would allow administrators, faculty, and student groups to reserve rooms for class meetings, meetings, and other functions on campus. The UMW community should be able to view a calendar that shows reserved rooms on any day. The calendar should note the times that each room is reserved and person who requested the reservation. Users should be able to search for available room based on room capacity (i.e. I need a room that will hold 30 or more people), building (i.e. I need a room in Trinkle Hall), date/time (i.e. I need a room on Tues Feb 8 from 3:00-5:00), AV equipment (i.e. I need a room with a projector), and seating (i.e. I need a room with tables not desks). The system should return a list of available room options and the user should be allowed to reserve a room from this list.

Now that you have a better understanding of the project, chat with a friend and try to identify some potential classes in the system that we missed on our first pass.

Identifying Responsibilities & Relationships

This section describes an approach for identifying classes their responsibilities and the relationships among the classes. This is a key component to creating a good solution to a complex problem.

Learning Objectives

- Understand how to use standard strategies to identify possible classes, responsibilities, and relationships in each project description.
- Understand the noun-verb walkthrough technique.

Last time we talked about discovering objects and classes. A rule of thumb is to read through the project description and look for __________.

When you start identifying classes and objects, you need to start thinking about how to distribute ________________________________ among them.

How to identify responsibilities

Rule of thumb: look for _____________words in the project description

This will help you to identify actions that the system needs to perform. You can assign these actions to the classes.

Example: We talked about Room being a potential class in the project. The primary responsibility of this class is to store information about a single Room at UMW. The data stored would include the building name, room number, seating capacity, and hasProjector (Boolean value).

Note: Even though responsibilities correlate with actions, they don't directly translate into methods. Each responsibility might be implemented through ________________ methods.

Can you think of multiple methods required to implement the Room class's responsibility?

Look at one of the items you flagged as a potential class in the room registration example from last class.   Identify 1-3 potential responsibilities for that class.

Identifying relationships – Dependencies, Aggregation, Inheritance

Dependencies, aggregation, and inheritance are three main types of relationships of interest to programmers. They are described in the following subsections.

Dependencies

Class1 is dependent on class2 if class1 _____________________ objects class2 in any way

Example: Suppose that a method in the Room class returns a String object. Makes the Room class dependent on the String class. In other words, if the String class is modified in the future, you might also have to modify your class because your method might not work correctly with the updated String class.

When you're designing classes, it's good to have few dependencies.

This is called ____.

Having a design with few dependencies makes it easier to update your classes later.

Aggregation

Class1 ____________________________ objects of class2
Aggregation is a special type of dependency

This is a "______________" relationship

Example: If you create a Patient class and inside the class you declare a data member of type Date. So, as a data member the Patient class **has a** Date object.

If the Date class changes, would you need to change the Patient class?

If the Patient class changes, would you need to change the Date class?

Inheritance

Class1 inherits ______________________________ from class2.

Class1 (the subclass) is a special case of the class2 (the superclass)

This is a "_____________" relationship

Example: Suppose you need to implement multiple types of users in your system: Staff and Students. You might implement a User class that stores information common to all users. Then you might inherit from the User class to create a specialized Staff class. You might also inherit from the User class to create a Student class.

If the User class changes, might you need to change the Staff class?

If the User class changes, might you need to change the Student class?

If the Student class changes, might you need to change the Staff class?

CRC Cards – A Tool to Discover Classes, Responsibilities, and Collaborators

Learning Objectives

- Given a project description, develop a list of potential classes
- Identify potential responsibilities for a given class
- Identify relationships that might be appropriate for a set of classes

CRC Cards

What are CRC cards? ____

On each index card describe ______________________________ and list its responsibilities and collaborators.

What are collaborators? ____________________________________

Classes that ____________________________________ in this class

Note: ____

The goal is to put down a bunch of ideas and then refine them. How might you refine them?

- If a class has too many responsibilities, ___________________________

Shoot for ______________________________ high-level responsibilities (which may turn into more than 3 methods at implementation).

- Eliminate classes with ____

- Remove responsibilities that do not ______________________________. Do not add extra responsibilities just because you can

CRC Cards in Early Design

How might you incorporate CRC cards into your early design?

Meet with a partner developer and examine the Project 1 Specification.

1. Do a noun/verb walk through
2. Create a candidate list for classes and operations
3. Develop user cases
4. Consider 1 use case
    - 1st person play protagonist & propose a responsible agent and method for the task
    - 2nd person play devil's advocate and ask for clarification or suggest an alternative
5. Put cards on table so classes are close to collaborators

As you talk through and review the classes, you can modify, discard, or create cards anytime.

Goal of CRC Cards

Remember, the goal of CRC cards is to discover classes, responsibilities, and relationships.

Review:

- Think of 3 questions that should be on next week's quiz related to the topics in the packet.

Functional, use cases, nouns, responsibilities, action, multiple, uses, loose coupling, instantiates, "has a", maybe, no, data and methods, "is a", note card (physical or electronic), classes, responsibilities, collaborators (relationships), one class, have dependencies, reduce its scope, 1 to 3, 0 responsibilities, add to final product,

Class 7 – Class Diagrams

CPSC 240 Tips for Good Design

Learning Objectives

- Understand Tips for Good Design

Tip 1: Modularize your code

A class represents an abstract _________________. Ideally the data type is a representation of ONE abstraction and has a _______________ responsibility.

Unless your project is incredibly simple, do not write all your code in a single class that contains only a main method.

Tip 2: Include a Javadoc header comment for every class

```
/**
   The Room class encapsulates data about a single room including the
   building name, room number, and seating capacity.
   @author Name Of Author
*/
```

Tip 3: Include a Javadoc method comment for every method

Provide a _______________ and use the @_________ and @_________ tags in your comments

```
/** Greet with a “Hello” message.
    @return message with “Hello” and name of greeted person
*/
String public sayHello( ) { ... }
```

Tip 4: Don’t use deprecated methods in the API. They are _____________________.

Tip 5: Include a constructor in your class

Why? This ensures that all objects have a _________________________ upon creation

Let’s look at an example class:
Example:

```
            class Room {
                  int roomNumber;
                  String buildingName;
                  int capacity;

            }
```

What should happen in a constructor?

What types of things should not happen in a constructor?

Tip 6: Preserve encapsulation

What is encapsulation? Grouping data/methods together into a new data type
Why? This prevents your class from being misused

For someone else to use your class, they do not need to know the details of your code; they just need to know the public interface.

Example:

```
            class Room {
                int roomNumber;
                String buildingName;
                int capacity;

            }
```

You may think that allowing direct manipulation from `main` keeps your program simple.

```
        Room aRoom = new Room();
        aRoom.roomNumber = 24;
        aRoom.buildingName = “Trinkle”
        aRoom.capacity = 100;
```

Fine, but what happens when if the Room author changes the data type of roomNumber changes to a String?   Will our code in main still work?

Tip 7: Don’t use I/O in your class if you can avoid it.

NOTE: Including a method to assist with debugging is okay.
Why? Hardcoding the spacing, labels, and other output details __________ the reusability of your code.

Example:
```
class Room {
   int roomNumber;
   String buildingName;
   int capacity;

   public void printRoom () {
      System.out.println(“\n\n” + buildingName + “ “ + roomNumber + “\n”);
   }
}
```

What happens if you need to print the room info in another format? What if you reuse this code later in another context and you don’t display information in the same way?

It may be a better idea to include accessor methods that allow you to customize how the data is displayed in main.
Example:
```
        Room aRoom = new Room();
        System.out.println(“The room you reserved is: “ +
                           aRoom.getGuildingName() + “:” +
                           aRoom.getRoomNumber() + “\n\n”);
```

UML Class Diagrams and CASE Tools

Learning Objectives

- Understand Class Diagrams and CASE tools

- Understand standard UML notation for representing classes including data members and methods as well as relationships between classes.

Class Diagrams

Goal: ____

Often drawn using a Computer Aided Software Engineering (CASE) tool, for

example: ____

Class is depicted as a _____________________________________

Example:

You can include the class name, attribute information and method information.

You can include all of these items, some of these items, or no details.

If you want to include the data member details, use this format
 visibility identifier: type

visibility

- + public
- - private
- # protected
- ~ package

Example:

```
-name: String
-age: int
```

If you want to include the method details, use this format
 visibility methodName(identifier: Type): returnTypeIdentifier

Example:

```
+getName() : String
```

Sometimes you do not include all information in your class diagram, which allows your class diagram to focus on the higher-level organization of your program and to save space.

You can depict relationships

Dependency

Aggregation

Inheritance

And many others – see the text book for details

You can depict multiplicity

Any number (0 or more)
1 or more
zero or 1

exactly 1

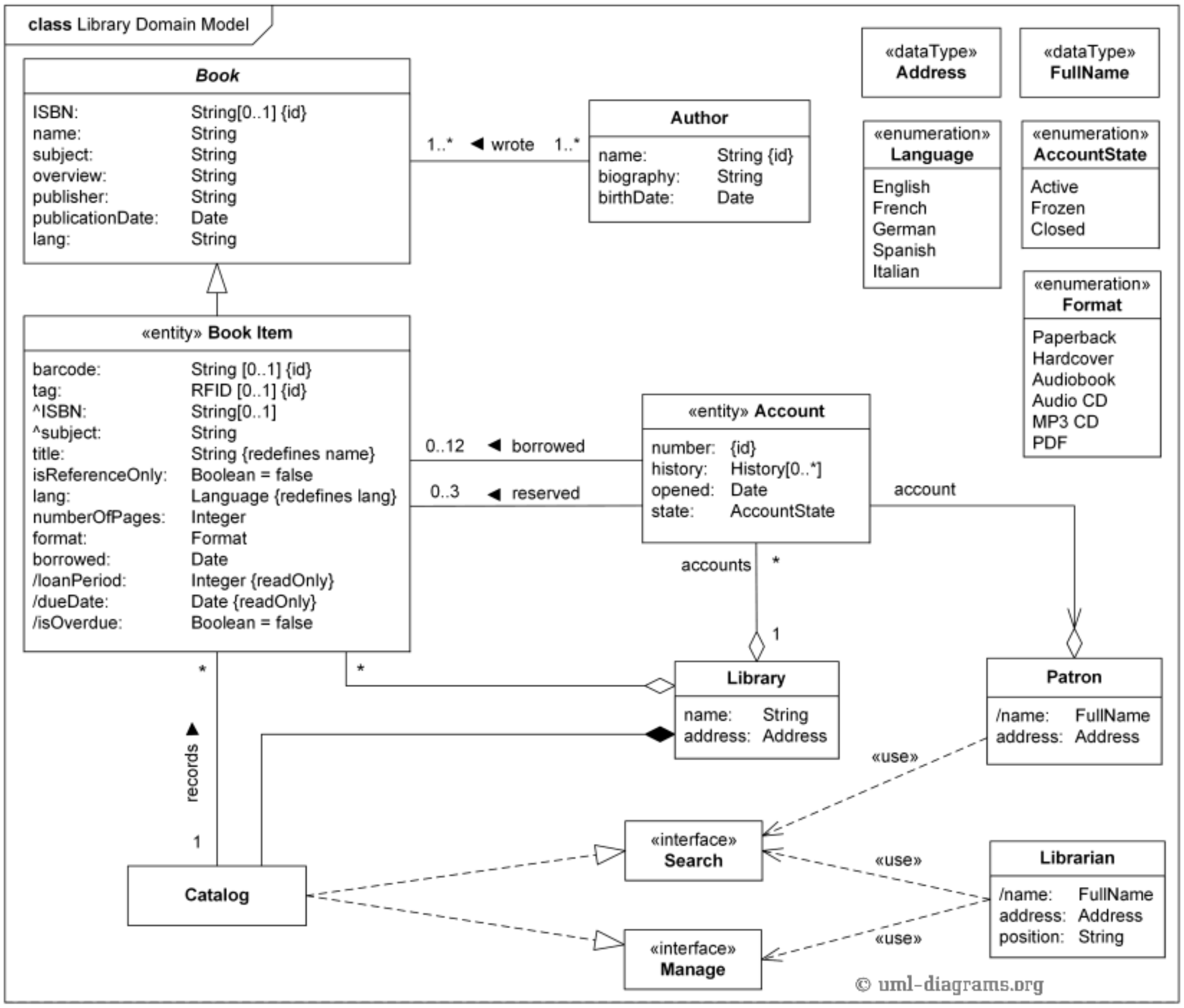
Example: This diagram shows that each message exists in exactly 1 message queue.   And each message queue holds zero or more messages.

You can also depict responsibilities by writing them on the line for clarification

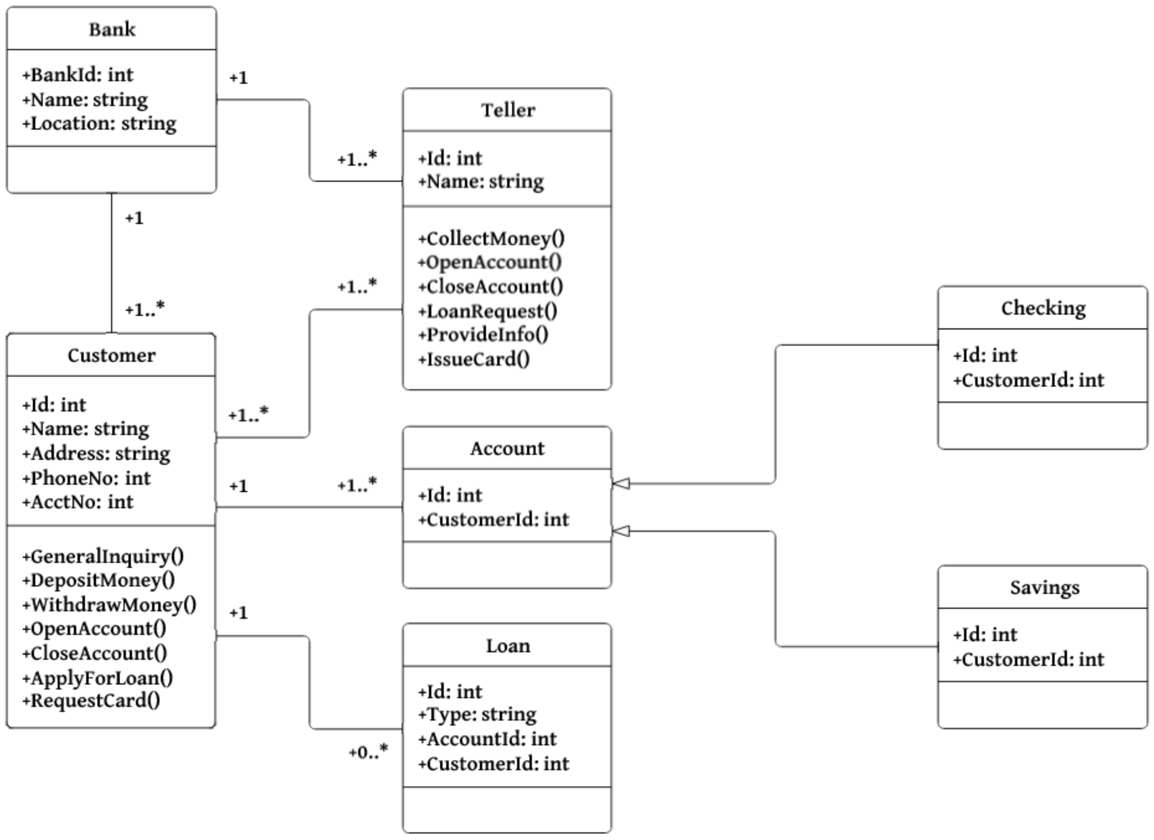
Example: A student registers for a course.   A course has a student participant.

Class Diagram Examples

Example from: http://www.uml-diagrams.org/examples/class-example-library-domain.png



Example from: https://d2slcw3kip6qmk.cloudfront.net/marketing/pages/chart/what-is-a-class-diagram-in-UML/UML_class_diagram_example2-750x539.PNG



Data type, single, valid initial state, description, param, return, not supported, valid initial state, initialize fields, I/O, limits, visual overview of problem solution, Visual Studio, rectangle with three areas,

Class 8 – SOLID Principles (T- 2/14)

Learning Objectives, Assignments, and Readings

- Understand the Single Responsibility Principle
- Understand why this principle is important in software development
- Understand how to identify code that violates the Single Responsibility Principle
- Write code that doesn't violate the Single Responsibility Principle

References

https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

Designing Classes

When writing classes we want to design for 5 things:

________________ – all methods are related to a single abstraction

_________________ – supports all operations that are part of the abstraction

Note: You need not implement every possible operation for your class to be complete. You want it to completely implement the abstraction is its primary responsibility, but not more than that.

_________________ – make it easy for someone else to use your class, especially the most common operations required
_________________ – clear to programmers without generating confusion

Example:

- Select good identifiers for your methods
- Include Javadoc comments so that people know what to expect from your methods in terms of parameters and return values
- In comments, clearly state your methods preconditions and postconditions

_________________ – operations should be consistent in terms of names, parameters, return values, and behavior
Example:

SOLID – The first letter – S

SOLID – An acronym for the 5 most agreed upon OO Design Ideas

To be an effective object oriented programmer, this is probably the single best thing that you could read and wrap your head around. Google it!

Note: SOLID is object-oriented not Java specific, so you will find write-ups using lots of other OO languages. The principles still apply to Java even though the syntax will be different.

S = Single Responsibility Principal
One class should have one and only one responsibility.
You should write your class for only one purpose.

WHY? Having a single responsibility will allow your class to be modified in the future without needing to worry about dependencies and how the changes may affect another part of your code.

Your class might model a real-world object and bundle data about it.
Example:

```
public class Room {
      String buildingName;
      String roomNumber;
      int capacity;
```

```
		public Room() { capacity = 0; }
		public Room(String bN, String rN, int c) {
			setBuildingName(bN);
			setRoomNumber(rN);
			setCapacity(c);
		}
		public String getBuildingName( ) { return  buildingName; }
		public String getRoomNumber( ) { return roomNumber; }
		public int getCapacity( ) { return capacity; }
		public void setBuildingName( String bN)  { buildingName = bN; }
		public void setRoomNumber (String rN) { roomNumber = rN; }
		public void setCapacity(int c) { capacity = c;}
	}
```

Example: Mailbox class (page 118 in OOD&P book)

```
public class Mailbox {
	…
	public void addMessage(Message aMessage) { … }
	public Message getCurrentMessage( ) { … }
	public Message removeCurrentMessage( ) {… }
	public void processCommand(String command) { … }
}
```

Do all the methods above relate to the same abstraction?

Example:
A typical example is a user management class. When you for instance create a new user you'll most likely send a welcome email. That's two reasons to change: To do something with the account management and to change the emailing procedure.

Example:

```
	class Book {
		String getTitle() { return “A Great Book”; }
		String getAuthor()  { return “John Doe”; }
		void turnPage( ) { //return a reference to the next page }
		void printCurrentPage( ) {
			System.out.println(currentPageContent);
		}
	}
```

Think about what type of classes might call the methods in this object.
The Book class might be used by a Data Presentation Mechanism (a GUI).
It might also be used by a Book Management class (like a librarian).

Would they both want the information about the book to be presented in the same way?

A better implementation for the Book class

```
      class Book {
            String getTitle() { return “A Great Book”; }
            String getAuthor()  { return “John Doe”; }
            void turnPage( ) { //return a reference to the next page }
            Page getCurrentPageContent( ) {
                  return(currentPageContent);
            }
      }

      interface Printer {
            void printPage(Page p);
      }

      class PlainTextPrinter implements Printer {
            void printPage(Page p) {
                  System.out.println(content of p);
            }
      }

      class FancyPrinter implements Printer {
            void printPage(Page p) {
                  System.out.println(“~~~~~~~~~~~~~~~~”);
                  System.out.println(content of p);
                  System.out.println(“~~~~~~~~~~~~~~~~”);
            }
      }
```

We will cover other SOLID principles through the semester.

Cohesion, completeness, convenient, clarity, consistency

Module 3 – Java OO Features

Module 3 Overview

In Module 2 we studied ways to identify OO features in a problem statement. In Module 3 we study Java's features that support OO. We also discuss JavaFX, Unit Testing, and Version Control.

Class Member Accessibility

Modifier	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
`public`	yes	yes	yes	yes	yes
`protected`	yes	yes	yes	yes	no
no modifier	yes	yes	yes	no	no
`private`	yes	no	no	no	no

Inheritance

We discuss inheritance in class 10. Other resources for inheritance:
http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
http://www.homeandlearn.co.uk/java/java_inheritance.html

Abstract Classes

We discuss Abstract Classes in Class 18, which is in Module 6, Design Patterns.

Generic Resources

http://docs.oracle.com/javase/tutorial/java/generics/

Lectures

- C10Inheritance.docx
- C11JavaFX.docx
- C12InterfaceTypes&Polymorphism.docx
- C13UnitTesting.docx
- C15VersionControl.docx

Labs

- Lab6Inheritance.docx
- Lab7JavaFX.docx
- Lab8Interfaces&Polymorphism.docx
- Lab9GroupQuestionnaire.docx
- Lab10UnitTesting.docx
- Lab11VersionControl.docx

Class 10 – Inheritance

Learning Objectives

- Understand the concept of inheritance
- Understand how to identify parent/child relationships (sometimes called superclass/subclass relationships)
- Given a parent class, understand how to implement a child class with specified additions
- Understand how to identify places where inheritance can't be used

Inheritance Overview

The big picture concept behind inheritance is ______________________________.
You have an existing class. You need to make a new class that has all the characteristics of the existing class and has either_________________________
and/or ___________________________associated with it.

EX: Which of the following would be the "base class" – the original class with the functionality/data that will be shared by all the classes?
User, AdministrativeUser

Employee, Manager

Student , UMWPerson, Faculty, Dean

Note: Another term for "base class" is "superclass." Use whichever term makes the most sense to you.
The terms for the class that extends the original are: "subclass", "child class" and "derived class." You can use these 3 terms interchangeably.

Syntax for Inheritance

```
public class Employee {
    private String name;
    private double salary;

    public Employee(String n) { name = n; }
    public void setSalary(double s) { salary = s; }
    public String getName() { return name; }
    public double getSalary() { return salary; }
}

public class Manager extends Employee {
    private double bonus;
    public Manager(String n, double b) { super(n); bonus = b; }
    public double getSalary() { ... } // overide Employee method
}
```

When inheriting from a superclass, you only declare the ________________________ between the subclass and superclass. The subclass automatically inherits ___________________________ and ____________________________ from the superclass.

How many data members does an object of type `Manager` have?

Which methods can be called by an object of type `Manager`?

Question: Is it possible to inherit just some selected data members and/or methods from the superclass?

Question: When implementing a subclass it possible to change the behavior of a method that is in the base class?

You can override a base class' method by writing a ______________________________ for that method ____in the superclass.

EX:

Remember the superclass is more _____________________. The subclass is more _________________.

Classes can be related through multiple levels of inheritance. This forms a ________________________.

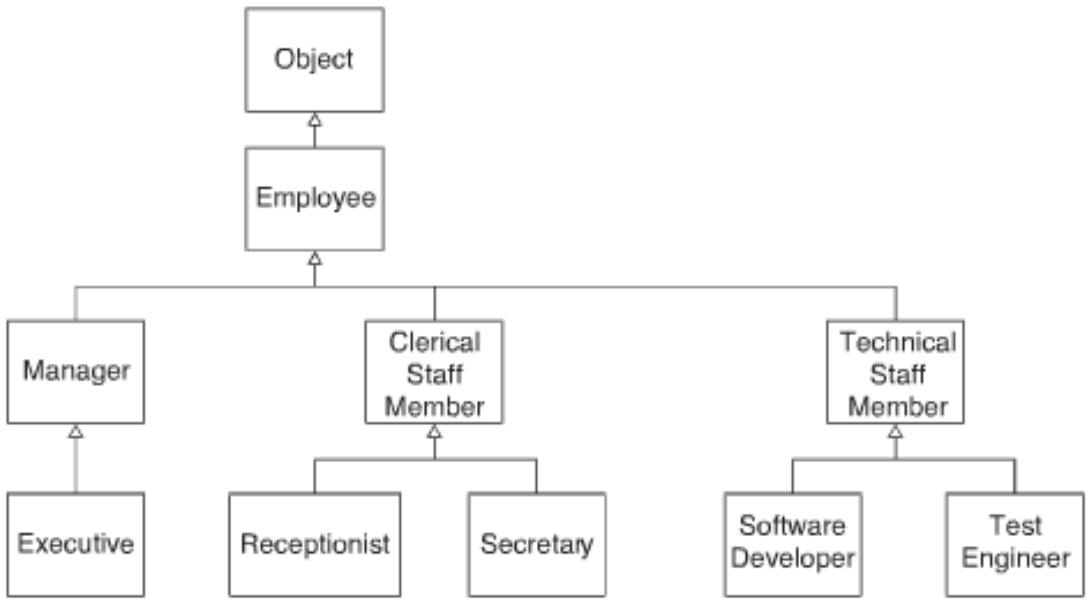


Figure 3

A Hierarchy of Employee Classes

Since a subclass "is a" superclass, you can substitute a subclass anywhere a superclass is expected. This is called the **Liskov ________________________________ principle**, where Liskov is the letter L of SOLID.

Example of Liskov:

```
Employee e;
…
e = new Manager("John Doe");
e.getName();  //calls method in the ______________class since the
____________________
e.getSalary(); //calls the ________________ version of the getSalary( ) method because of polymorphism
```

Let's look at how we'd implement the `getSalary( )` method in the `Manager` class

```
public double getSalary( ) {
    return salary + bonus; //ERROR:
}
```

What about:

```
public double getSalary() {
    return getSalary()+ bonus; //ERROR:
}
```

Solution

```
public double getSalary( ) {
    return  ____________________ ;
}
```

Another option: use ____________________________ rather than `private` variables in your superclass.

The protected visibility level means that the class member is accessible within any code that extends the original class. But it's not accessible from outside the inheritance chain.

Suppose that the Clerical Staff Member class (from the hierarchy on the previous page) has a data member:

```
protected String faxNumber;
```

Is this data member accessible in each of the following classes?
Clerical Staff Member?
Employee?
Secretary?
Executive?

Class Member Accessibility

Oracle describes controlling access to class members at the following site.
https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

The following table summarizes access.

Modifier	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
public	yes	yes	yes	yes	yes
protected	yes	yes	yes	yes	no
no modifier	yes	yes	yes	no	no
private	yes	no	no	no	no

Reusability and extending code, additional data, new methods, User, Employee, UMWPerson, differences, all the data, all the methods, three, all in Manager and Employee, No, Yes, new implementation, signature must be the same, broad, specific, hierarchy, substitution, Employee, method does not exist locally, Manager, super.getSalary() + bonus, protected, yes, no, yes, no

Class 11 - JavaFX

Learning Objectives, Assignments, and Readings

- Understand the basics of GUI programming.

- Understand how to build an FXML-based JavaFX application, including the Model-View-Controller aspects.
    - Understand the view portion is in a .fxml file.
    - Understand the controller portion is in its own .java file.
    - Understand the business portion is in its own collection of .java files.
- Understand how to use SceneBuilder to drag and drop GUI components on a Window.
    - Understand how to name variables and handler methods in SceneBuilder.
    - Understand how to use SceneBuilder to edit a .fxml file.
    - Understand how to use SceneBuilder to extract code for the controller

JavaFX

JavaFX is Java's third generation Graphical User Interface system. Java created Abstract Windowing Toolkit (AWT) its first GUI library in 1995. AWT GUI components are rendered and controlled by the GUI components of the underlying OS. Java created Swing its second GUI library in 1997. Swing is built on top of AWT, and building a Swing GUI application often uses features from both AWT and Swing. In 2008, Java introduced JavaFX and it has several releases. JavaFX is intended to replace Swing, but both will remain for the foreseeable future[2]. JavaFX and Swing are similar, but we will use JavaFX as part of our class, labs, and projects. We will not become expert GUI developers. We will use GUI components just enough to have a basic user interface to our applications.

Scene Builder

Scene Builder is a drag-n-drop GUI builder for creating .fxml files. A .fxml file is the following.

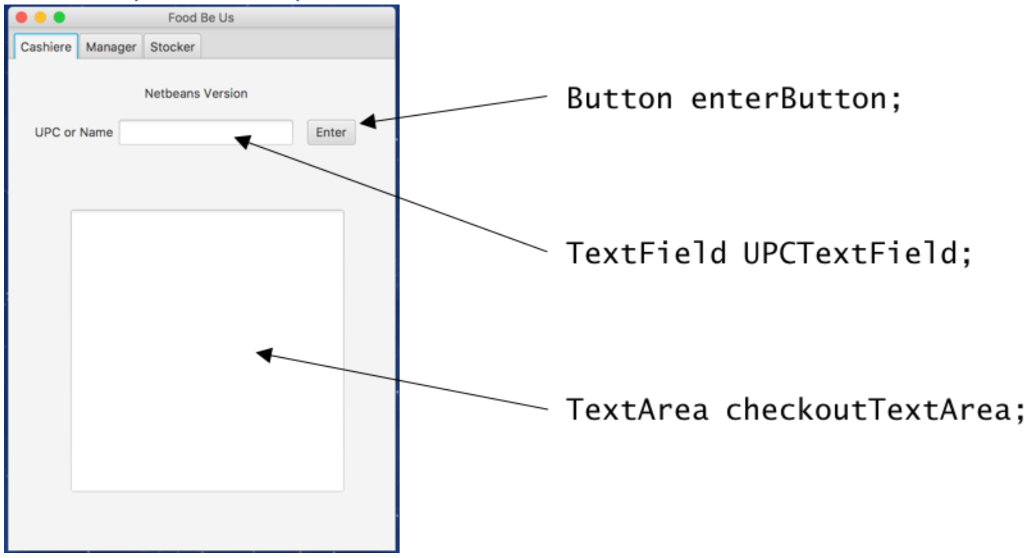
- A text file that looks like XML.
- describes the GUI components on a JavaFX program.
- Is loaded by FXMLLoader when a JavaFX program begins.

Scene Builder is free. It is maintained by Gluon. Scene Builder can be downloaded at the following link.

http://gluonhq.com/products/scene-builder/

[2] See http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html

Three Simple GUI Components



This section describes three simple GUI components and how we can use them in our Java programs. Our hands-on JavaFX lab demonstrates using these concepts. Each GUI component allows us to do the following.

- Connect a method to the component that is called on specified events. The sample diagram connects `handleEnterButton(MouseEvent event)`, to `enterButton`. The method is called when a mouse clicks in button.
- Get text from TextField and TextArea components. Getting text is typically done when a user clicks on a button and hits the enter key. In the sample diagram, the method `handleEnterButton` calls `UPCTextField.getText()` to get what user has typed.
- After processing the typed information, the resulting output is placed in the `TextArea` by calling `checkoutTextArea.setText()`. You can also call `UPCTextField.setText()`.

Class 12 – Interface Types and Polymorphism

Learning Objectives, Assignments, and Readings

- Understand the concept of polymorphism
- Understand implications of polymorphism for object oriented code
- Explain how interfaces are useful in designing large programs

Polymorphism

Polymorphism is the ability to select the appropriate method for an object based on context.

Today we explore this concept and why it is helpful in programs.

Collections Class with Comparable Interface Example

Consider the `Collections` class in the Java library. The `Collections` class contains a static method called `sort()` which can sort an array list

```
    Collections.sort(list);
```

The objects in the list can be of any type if the type implements the `Comparable` interface type.

This allows the program to be sure that ______________________________________

EX:
```
public interface Comparable<T> {
    int compareTo(T  other);
}
```

So, what is an interface type anyway?

Interface: ____

It's like a stub that provides some information about a type, but allows you (the interface user) the flexibility of making implementation choices.

An interface allows you to create a ________________________ for a datatype. You define the methods that must be included, but don't include __________________ or __________________________________.

Obviously, you cannot use this sort of thing on its own, but you can use it as a template to design a class that fits with some existing code.

You can write multiple classes that can implement the same interface – each implementing its operations in its own way. This helps to make code _____________________________.

The `Comparable` interface is a generic type.   When you define the interface, specify the type to be used.

EX:
```
Comparable<Student> aStudent;
Comparable<Student> anotherStudent;
//then you can say
 if (aStudent.compareTo(anotherStudent) > 0)
        //swap position of aStudent and anotherStudent in list
```

Just FYI, `compareTo()` is set up to return a negative number if the calling object should come before the parameter object, zero if they are equal, and a positive number if the calling object should come after the parameter object. (This is a standard programming convention for this type of comparison method.)

Having this type of flexible interface associated with the task of sorting is convenient because sorting always involves a comparison like the one shown above.   If your data type implements the Comparable interface, then it will play well with the sorting processes that are already written in the Java library. What does it look like when you write a class that implements an interface?

```
public class Student implements Comparable<Student> {
    private String name;
    private int ID;
    public Student (String aName, int anID) {
        name = aName;
        ID = anID;
    }
    public String getName( ) { return name; }
    public int getID( ) { return ID; }
    public int compareTo(Student other) {
        return ID - otherID;
    }
}
```

How to use

```
import java.util.*;
public class StudentSortTester {
    public static void main (String [] args) {
        ArrayList<Student> students = new ArrayList<Student>();
        students.add(new Student ("Seinfeld", 456789));
        students.add(new Student("Costanza", 123888));
        students.add(new Student("Kramer", 378931));
        Collections.sort(students);

        for (Student s : students)
            System.out.println(s.getName() + " "+ s.getID());
    }
}
```

Great but what if you want to change to sort by name instead of ID? You could change the `compareTo()` method implementation – but who wants to change their code all the time?

Collections Class with Comparator Interface Example

Another option is to use a different `sort()` method that does a different type of comparison. Implement the Comparator interface type, which requires one method

```
public interface Comparator {
      int compare(T first, T second);
}
```

Again, it will return a negative number, zero, or positive number

If `thingToCompare` is an object of a class that implements the Comparator interface type, then

```
Collections.sort(list, thingToCompare);
```

sorts the objects in the list according to the sort order that thingToCompare defines.

Example of Comparator Interface:
```
public class StudentComparatorByName implements Comparator<Student> {
      public int compare(Student s1, Student s2) {
            return s1.getName( ).compareTo(s2.getName());
      }
}
```

Then in `main()`, call

```
Comparator<Student> studentToComp = new StudentComparatorByName( );
Collections.sort(students, studentToComp);
```

Bonus: now you can use any type of object in the list (not just ones that implement the Comparable type).   Why might that be handy?

You could even write the comparator method so that it behaves differently based on a parameter.

```
public class StudentComparator implements Comparator<Student> {
public int StudentComparator(bool ascending) {
            if (ascending) direction =1;
            else direction = -1;
      }
      public int compare(Student s1, Student 2s) {
            return direction = s1.getName( ).compareTo(s2.getName( ))
      }
private int direction;
}
```

Then change how you call to change the sort direction:

```
//sort Z to A
      Comparator<Student> reverseComp = new StudentComparator(false);
//sort A to Z
      Comparator<Student> comp = new StudentComparator(true);
```

Highlights about interfaces

- Interfaces never have instance variables.   Interfaces can have named constants (declared as static final and provided with a value at declaration).
- A class can implement as many interfaces as you want it to.
  EX: `public class MarsIcon implements Icon, Shape { ….`
- An interface can extend another interface

  ```
  public interface MoveableIcon extends Icon {
      void translate(int x, int y);
  }
  ```

Classes that implement `MoveableIcon` will have to implement `translate()` and all the methods defined by the Icon interface.

Compare two elements, guarantees your class includes methods with signature defined in interface, set of expectations, method implantation, variable declarations, compatible, create your own sorting order

Class 13 – Unit Testing

Learning Objectives, Assignments, and Readings

- Understand testing and various testing types.
- Understand unit testing.
- Understand testing frameworks.
- Understand JUnit.

Testing

Testing software is just as important as implementing it. Recall the software development process has an implementation phase and a testing phase. Also recall that in an agile software development process there is an oscillation between implementation and testing. The various types of testing that occurs are described in the following subsections.

Unit Testing

Unit testing tests an individual unit of code. In Java, you perform unit testing on a _________ __________, by constructing objects and calling all methods in the class. Often a method requires several __________ ___________ to test it. This class focused on unit testing.

Integration Testing

Integration testing tests the entire software system. All classes that comprise the system are built into an executable image, which is tested. Integration testing is typically performed from a ___________ perspective.

Code Coverage Testing

Code coverage testing runs test cases and collects _____________ about the various lines of code that have been executed. Code coverage testing can include unit and integration tests.

Performance Testing

Performance testing measure the performance of software. Performance testing includes such efforts as determining ___________ ____________ of the algorithm, how much memory code uses, and how much file I/O the code performs. The results from performance testing can be used to provide system requirements for running the software.

Test Driven Development

Test driven development is a ______________ ________________ _____________ where you oscillate between programming a class and programming the unit test cases for the class. IntelliJ and NetBeans support this oscillation between your classes and JUnit classes.

Java Unit Testing

JUnit

JUnit is a testing framework for performing unit testing on Java classes. JUnit is open source host on GitHub – https://github.com/junit-team/junit5. The testing framework consists of the following.

- JUnit Package – A package of classes that can be used to construct tests.
- Annotations – A collection of annotations used to annotate methods. The most used annotation is ________________ that identifies a method as a test case.
```
@Test
void getSalePrice() {
   BikePart bp1 = new BikePart("saddle1","1234567890","45.49", "40.00",
                                     false);
   assertEquals("40.00", bp1.getSalePrice());
}
```
- Assertion Methods – A collection of assertion methods that can be used to assert ___________ ___________ that are compared to the ____________ _____________.
- Test Runner – A mechanism to run the tests. IntelliJ, NetBeans, and Eclipse have built in test runners. You can also run JUnit from the command line.

IntelliJ JUnit Testing

The following video shows the basics of using JUnit with IntelliJ.
https://www.youtube.com/watch?v=Bld3644bIAo

Netbeans JUnit Testing

The following video shows the basics of using JUnit with NetBeans.
https://www.youtube.com/watch?v=Q0ue-T0Z6Zs

JUnit Annotations

- `@test` – Identifies a test method.
- `@test (expected = Exception.class)` – Fails if the method does not throw the named exception.
- `@test (timeout = 100)` – Fails if the method takes longer than 100ms.
- `@BeforeClass` – Static methods are executed one time before that start of tests.
- `@AfterClass` – Static methods are executed one time after all tests.
- `@Before` – Method is executed before each test case. For example, you may open a file and read some information.
- `@After` – Method is executed after each test case. For example, you may close a file.

```
@Before
public static void setUp(){}
@BeforeClass
public static void setUpClass() throws Exception {}
@AfterClass
public static void tearDownClass() throws Exception {}
```

JUnit Assertions

In each of the following assertions, message is an optional message displayed when the assertion fails.

- `assertEquals([message,] expected, actual)` – check expected and actual are same value.
- `assertEquals([message,] expected, actual, tolerance)` – check expected float/double and actual float/double are same to a tolerance of number of decimal digits.
- `assertFalse([message,] boolean)` – check boolean expression if false.
- `assertTrue([message,] boolean)` – check boolean expression is true.
- `assertNull([message,] object)` – compares object to null
- `assertNotNull([message,], object)` – compares object to not null
- `assertSame([message,] expected, actual)` – checks expected and actual refer to same object
- `assertNotSame([message,] expected, actual)` – checks expected and actual refer to differen objects

Pre-conditions and Post-conditions

Pre-conditions and post-conditions are assertions, that are sort of related to the JUnit assertions; however, pre-conditions and post-conditions use the Java `assert` statement. Pre-conditions tell what is expected of parameters when a method is called. Post-conditions check what must true when the method is finish. In both cases, when the `assert` statement fails, it throws an `AssertionError`. The Java `assert` statement can be turned-off at compile time. This means they will not be included in the resulting code. For this reason, Java does not recommend using assertions on parameters of public methods. For public methods that check parameters, Java recommends that you explicitly check parameters and throw your own exceptions

A pre-condition states what must be _____________ of the parameters when a method is called for the method to operate correctly. If the precondition is not true when the method is called, then your code makes ______________________________ about the outcome.

A post-condition states what must be _________________ after the method ends, assuming the preconditions were met when the method is called.

EX:

```
private int size = board.length;
/**
   Update the position of the player to a new row/column.
   @param row  The new row location of the player
   @param column  The new column location of the player
   Precondition: The row and column params are valid
   Postcondition: Player position updated to valid row and column



*/
void updateLocation(int row, int column) {
        assert ((row < size) && (column < size) &&
                (row >=0)    && (column >=0));
        playerPositionRow = row;
        playerPositionColumn = column;
}
```

Syntax:

```
        assert (condition)
```

If the condition is true, execution continues.
If the condition is false, an `AssertionError` is thrown and execution stops.

Assertions can be turned on/off ___________________________.
Assertions are primarily helpful during debugging.

single class, test cases, user, statistics, execution speed, software development process, @test, expected values, actual values, true, no guarantee, true

Class 15 – Version Control

Learning Objectives, Assignments, and Readings

- Understand version control.
- Understand Git.
- Understand GitHub.

Version Control

Version control is keeping track of various versions of your software. Everyone has updated their phone apps to newer versions. When Facebook, or any app, fixes bugs and/or adds new features to their software, they create a new version. You can create versions during your development. Suppose you are developing a Bicycle Warehouse program, and your first goal is to simply load the main warehouse with bicycle parts from a warehouse delivery file. You could implement these capabilities and create a version before adding new features. A version control tool allows you to select the code from any of its saved versions. Being able to revert to a previous version of your software is a convenient feature. The following are a couple of examples.

- If you are adding new capabilities and accidently delete good lines of code, you use the version control tool to retrieve the previous version that had those lines of code.
- If you are testing and discover a bug that was not present earlier in your development, you can examine previous versions to see when you introduced the bug.

Manual Version Control

Suppose you have a software project in a folder – `MyBicycleWareHouse`. This folder may have subfolders that represent packages, documentation, and input/output files. Suppose you have just finished developing and testing Version 1.0. You could manually perform version control by copying `MyBicycleWareHouse` and all subfolders to a version 1.0 folder. For example, using a Linux recursive copy command.

```
$ cp –R MyBicycleWareHouse MyBicycleWareHouseVersion1.0
```

After this command, the folder `MyBicycleWareHouseVersion1.0` contains version 1.0 of your software project. You can now develop Version 2.0 in the original `MyBicycleWareHouse` folder.

Git - a shortened version of http://gitref.org

Git is a version control tool that stores snapshots of our project, where a snapshot is like copying the entire project; however, the project is not really copied. Instead, Git records a manifest of what your project files look like at the point the snapshot is taken. The manifest is stored in the .git folder. The following discussion uses Linux commands to demonstrate the concepts; however, you will perform them using your IDE. For example, instead of typing `git commit` you may select a button to perform the commit, which will open an edit window for you message.

Git – Creating Git Repository

Suppose Suzy has a folder, `gitPractice`, in which she has a few files – `README`, `BikePart.java`, `BikePartTester.java`. You create the Git repository using the `git init` command.

```
$ pwd
gitPractice
$ ls
BikePart.java     BikePartTester.java     README
$ git init
$ git config –global user.name “Suzy”
$ git config –global user.email “suzy@email.com”
$ ls –a
.                 .git                BikePartTester.java
..                BikePart.java       README
```

If you examine the status of your Git repository at this point, you will see you have three files that are not being tracked.

```
$ git status –s
?? BikePart.java
?? BikePartTester.java
?? README
```

You must add the files to your Git repository for Git to begin tracking them.

Git – Adding and Committing Files to Git Repository

A Git repository has a two-stage process for placing files in it. First files are added to the repository using `git add` and then files are committed to the repository using `git commit`. When you commit files to the repository, you include a message describing the commit. The example commit shown uses the `-m 'message here'`, but `git commit` allows entering a message (comment) via an editor when the `-m` is omitted.

```
$ git add README BikePart.java BikePartTester.java
```

Alternatively, you can add all files in a folder with the following.

```
$ git add .
```

You can see the status of your repository with `git status -s`.

```
$ git status -s
A BikePart.java
A BikePartTester.java
A README
```

You can also examine status without the –s.

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

	new file:   BikePart.java
	new file:   BikePartTester.java
	new file:   README
```

To complete the second stage, you perform the following command.

```
$ git commit -m 'Demonstration of Git'
[master (root-commit) d4e0a98] Demonstration of Git
 3 files changed, 11 insertions(+)
 create mode 100644 BikePart.java
 create mode 100644 BikePartTester.java
 create mode 100644 README
```

Git, Remote Repositories, and GitHub

We want to store our local Git repository on a remote Git repository, and GitHub[3] (https://github.com) is where we will store it. GitHub provides free storage of public repositories, which means they are open for anyone to view. If you were a company, you would pay for a private repository.

In this scenario, there are two teammates who will use one remote GitHub repository to work on a common project. Both teammates create an GitHub accounts, and one of the teammates creates

[3] There are other alternatives such a BitBucket, but we will use GitHub.

a repository in which to store the project code. The a GitHub repository that parallels the local Git repositorys. The teammates use `git push` and `git pull` commands to move data between local repositories and the Github repository.

Teammate Suzy created the initial code as described in the previous sections. Suzy's local Git repository is in `gitPractice` folder on her computer. Suzy's GitHub account is suzy, and she created a BikePart repository. Suzy's BikePart repository URL is https://github.com/suzy/BikePart.git. Suzy connects her GitHub repository to her local repository as follows.

```
$ git remote add origin https://github.com/suzy/BikePart
$ git remote -v
origin      https://github.com/suzy/BikePart (fetch)
origin      https://github.com/suzy/BikePart (push)
```

In this example, `origin` is the name of the remote repository that is used by git commands. Origin is a typical Git name for the remote repository. We could have named the remote repository something else as follows.

```
$ git remote add ongithub https://github.com/suzy/BikePart
```

Now we can push and fetch from the local Git repository to the GitHub Git repository. If I attempt to push at this point, I get an error because my GitHub repository has a README file that is not in my local repository.

```
$ git push origin master
```

At this point, your GitHub repository has the files that were in your local Git repository. Your teammates can now clone the GitHub repository onto their computers. They can create a folder cd into the folder, use the `git clone` command, which creates a local Git repository with BikePart.java and BikePartTester.java.

```
$ mkdir gitPractice2
$ cd gitPractice2
$ git clone https://github.com/suzy/BikePart .
$ ls
BikePart.java           BikePartTester.java     README
```

You can see that this remote is pointing to the same GitHub repository as previous.

```
$ git remote -v
origin      https://github.com/gustycooper/BikePart (fetch)
origin      https://github.com/gustycooper/BikePart (push)
```

The teammate can make changes and push them to GitHub. I edit BikePart.java and perform the git status.

```
$ git status
```

```
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

      modified:   BikePart.java

no changes added to commit (use "git add" and/or "git commit -a")
$ git commit -a
[master ee28c5f] Updated BikePart.java from a teammate account.
 1 file changed, 1 insertion(+)
$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 397 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/gustycooper/BikePart
   4ae79dd..ee28c5f  master -> master
```

Now I can return to the original `gitPractice` folder and perform a `git pull origin master`, which will update the files to match those on GitHub. Notice `BikePart.java` has changed.

```
$ cd ../gitPractice
git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/gustycooper/BikePart
 * branch            master     -> FETCH_HEAD
   4ae79dd..ee28c5f  master     -> origin/master
Updating 4ae79dd..ee28c5f
Fast-forward
 BikePart.java | 1 +
 1 file changed, 1 insertion(+)
```

Now, suppose the teammate working in `gitPractice` adds a new file, `LoginAccount.java`.

```
$ git add LoginAccount.java
$ git commit -m ‘Added LoginAccount to project’
$ git push origin master
```

The teammate working in gitPractice2 can perform a git pull to get the latest version of files on the GitHub repository.

```
$ cd ../gitPractice2
$ git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
```

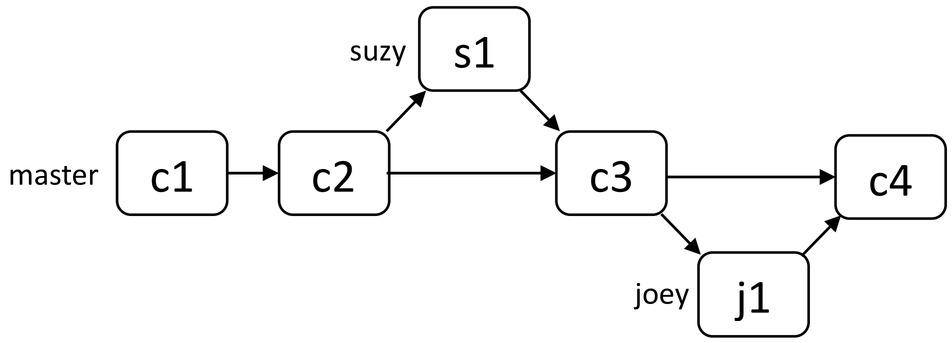
```
Unpacking objects: 100% (3/3), done.
From https://github.com/gustycooper/BikePart
 * branch            master     -> FETCH_HEAD
   ee28c5f..b940545  master     -> origin/master
Updating ee28c5f..b940545
Fast-forward
 LoginAccount.java | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 LoginAccount.java
$ ls
BikePart.java           BikePartTester.java     LoginAccount.java README
```

Git Use and Teammate Coordination

The previous sections describe a way that teammates can use Git and GitHub to develop a project. This is a simple way, but it requires coordination between teammates. The approach works best when teammates do not update the same files concurrently. The team would divvy work such that teammates are working on different files. They could message each other when their files are ready for testing, allowing teammates to perform a git pull to get the latest files.

Git Branches

In this scenario, Suzy (in gitPractice) creates a branch names suzy; and Joey (in gitPractice2) creates a branch named joey. First Suzy edits BikePart.java on her branch, and then Joey edits BikePartTester.java in his branch. Suzy and Joey push their branches to GitHub, which results in three branches – master, suzy, and joey. At GitHub we perform one pull request to merge suzy branch into master and a second pull request to merge joey into master. We also delete suzy and joey from GitHub. GitHub's master has changes from both Suzy and Joey. Suzy and Joey then pull master into their local Git repositories and continue working. The following figure demonstrates this concept.



Suzy's Local Git Branch - suzy

```
$ git branch suzy
$ ls
BikePart.java           BikePartTester.java     LoginAccount.java README
$ git branch
* master
  suzy
$ git checkout suzy
Switched to branch 'suzy'
```

```
$ git status
On branch suzy
nothing to commit, working tree clean
$ ls
BikePart.java           BikePartTester.java     LoginAccount.java README
$ vim BikePart.java
$ git status
On branch suzy
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

      modified:   BikePart.java

no changes added to commit (use "git add" and/or "git commit -a")
$ git commit -a
[suzy 2b9f71c] Changed BikePart.java on suzy branch.
 1 file changed, 1 insertion(+)
$ ls
BikePart.java           BikePartTester.java     LoginAccount.java README
$ git status
On branch suzy
nothing to commit, working tree clean
```

Pushing Suzy's Local Git Branch to GitHub

```
$ git push origin suzy:suzy
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 354 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/gustycooper/BikePart
 * [new branch]      suzy -> suzy
```

Joey's Git Branch

At this point, the GitHub repository has two branches, master and suzy. The following is Joey changing BikePartx.java.

```
$ cd ../gitPractice2
$ ls
BikePart.java           BikePartTester.java     LoginAccount.java README
$ git branch joey
$ gits status
-bash: gits: command not found
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
$ git checkout joey
M     BikePartTester.java
Switched to branch 'joey'
$ vim BikePartTester.java
$ git status
```

```
On branch joey
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

	modified:   BikePartTester.java

no changes added to commit (use "git add" and/or "git commit -a")
Gustys-iMac:gitPractice2 gusty$ git commit -a
[joey 11b7c62] Changed BikePartTester.java on joey branch.
 1 file changed, 1 insertion(+)
$ ls
BikePart.java           BikePartTester.java     LoginAccount.java README
$ git status
On branch joey
nothing to commit, working tree clean
```

Pushing Joey's Local Git Branch to GitHub

```
$ git push origin joey:joey
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 333 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/gustycooper/BikePart
 * [new branch]      joey -> joey
```

Merging Branches on GitHub

At this point, the GitHub repository has two branches, master, suzy, and joey. You can select the suzy branch on GitHub and select create pull request. This should say suzy and master can be merged without conflicts. Once the pull request is done, you will have Pull requests 1 on the top navigation bar of your GitHub repository. You can select the joey branch on GitHub and select create pull request. This should say joey and master can be merged without conflicts. Once the pull request is done, you will have Pull requests 2 on the top navigation bar of your GitHub repository.

Select the Pull request 2 option on the top navigation bar, which shows a page with describing the two branches. Select the "Changed Bikepart.java on suzy branch." Perform the pull request, which will merge the suzy branch onto the master branch. After successful merge, select Delete Branch to delete the suzy branch. Repeat this with joey branch.

At this point the GitHub repository is a single branch – master – that has the updates that Suzy and Joey made. NOTE: Suzy and Joey worked on different files. If they had edited the same file, GitHub can still perform the merge if their edits did not conflict with each. If the edits conflicted, you would have to decide which of the edits to keep/discard.

Suzy Updating Her Local Git Repository

At this point, Suzy and Joey want to get the latest version of master into their local repositories. Suzy does this by the following.

```
$ cd ../gitPractice
$ ls
BikePart.java          BikePartTester.java     LoginAccount.java README
$ git status
On branch suzy
nothing to commit, working tree clean
$ git checkout master
Switched to branch 'master'
$ git branch
* master
  suzy
$ git pull origin master
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 3), reused 3 (delta 2), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://github.com/gustycooper/BikePart
 * branch            master     -> FETCH_HEAD
   b940545..71e41f9  master     -> origin/master
Updating b940545..71e41f9
Fast-forward
 BikePart.java       | 1 +
 BikePartTester.java | 1 +
 2 files changed, 2 insertions(+)
$ git branch -d suzy
Deleted branch suzy (was 2b9f71c).
```

Suzy Updating Her Local Git Repository

Joey gets the latest version of the GitHub master into his local repository as follows.

```
$ cd ../gitPractice2
$ ls
BikePart.java          BikePartTester.java     LoginAccount.java README
$ git status
On branch joey
nothing to commit, working tree clean
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
$ git branch
  joey
* master
$ git pull origin master
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 2), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://github.com/gustycooper/BikePart
 * branch            master     -> FETCH_HEAD
   b940545..71e41f9  master     -> origin/master
Updating b940545..71e41f9
Fast-forward
 BikePart.java       | 1 +
 BikePartTester.java | 1 +
 2 files changed, 2 insertions(+)
$ git branch -d joey
Deleted branch joey (was 11b7c62).
```

GitHub and .gitignore

Often you have more files in your local git repository than you want to store on GitHub. The `.gitignore` file is used to define which files are **NOT** pushed from your local repository to GitHub. Your IDE has pattern `.gitignore` files. The following is some of a `.gitignore` that I have for a Java project developed in IntelliJ.

```
# Package Files #
*.jar
*.war
*.ear
*.zip
*.tar.gz
# .gitignore
.gitignore
# IntelliJ folders
.idea/
out/
```

IntelliJ, Git, and GitHub

Using Git Integration with IntelliJ
https://www.jetbrains.com/help/idea/2017.1/using-git-integration.html#d465818e14
Setting up a local repository
https://www.jetbrains.com/help/idea/2017.1/setting-up-a-local-git-repository.html
Adding files to a local repository.
https://www.jetbrains.com/help/idea/2017.1/adding-files-to-a-local-git-repository.html
Committing changes to a local repository.
https://www.jetbrains.com/help/idea/2017.1/committing-changes-to-a-local-git-repository.html
Registering you GitHub account with IntelliJ
https://www.jetbrains.com/help/idea/2017.1/registering-github-account-in-intellij-idea.html
NOTE: I used the recommended Token approach.
Managing remotes – GitHub
https://www.jetbrains.com/help/idea/2017.1/managing-remotes.html
Push Dialog
https://www.jetbrains.com/help/idea/2017.1/push-dialog-mercurial-git.html

Module 4 – Design Patterns

Module 4 Overview

Design patterns are an important concept for programmers. A design pattern provides a best practice solution to well-known problems. We study an overview of design patterns along with a detailed analysis of several design patterns such as the following.

- Strategy Pattern
- Observer Pattern
- Model View Controller Pattern
- Template Pattern
- Decorator Pattern

References

The following are references for topics related to working in groups and design patterns.

Google's Unconscious Bias Video
https://www.youtube.com/watch?v=_KfKmGb_bT4

Tips for working in a group
https://www.cs.cmu.edu/~pausch/Randy/tipoForGroups.html

Pair Programming
https://www.youtube.com/watch?v=YhV4TaZaB84

Interview w/UMW Alum, Lucy Bain – two parts
https://www.youtube.com/watch?v=cl1PTUQvcX0
https://www.youtube.com/watch?v=bM0qnbNbIuM

Atlassian Pair Programming (Lucy Bain)
https://www.youtube.com/watch?v=fQ-x-T34z9w

Abstract Classes and Interfaces
https://www.youtube.com/watch?v=AU07jJc_qMQ

Why/When to Use Abstract Classes
https://www.youtube.com/watch?v=yyU3bXyc_oU

Template Method Pattern
https://www.youtube.com/watch?v=aR1B8MlwbRI

Lectures

- C16DesignPatterns&StrategyPattern.docx
- C17ObserverPatternMVC.docx
- C18AbstractClasses&TemplatePattern.docx
- C19DecoratorPattern.docx

Labs

- Lab12ObserverPattern.docx
- Lab13TemplatePattern.docx
- Lab14IteratorPattern.docx

Class 16 – Design Patterns and Strategy Pattern

Learning Objectives

- Understand design patterns
- Understand why patterns are useful tools for programmers
- Understand the standard format for design patterns

Introduction to Design Patterns

What is a design pattern?

____

They were originally document by a Gamma, Helm, Johnson, & Vlissides in their book Design Patterns, which is commonly referred to as the "___________________________" in CPSC circles.

The original book contains ____________ patterns.

Why should you know design patterns?

They've been proven over time to be efficient and effective for solving

____

It gives you a ____ that you can use to talk about coding problems & solutions

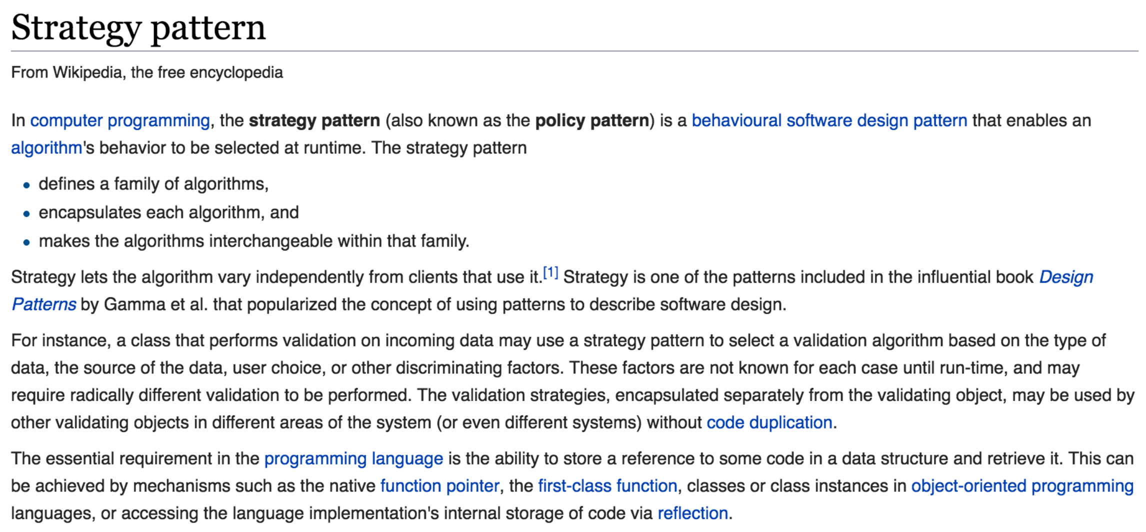
We're going to talk about a subset of these common design patterns:

- Strategy pattern
- Template Method Pattern
- Iterator pattern
- Observer pattern
- Decorator pattern
- Singleton pattern

Learning Objectives, Assignments, and Readings

- Understand what a design pattern write up looks like
- Understand the essentials of the Strategy Pattern
- Understand and recognize the Strategy Pattern in project descriptions

Strategy Pattern from Wikipedia



Strategy pattern

From Wikipedia, the free encyclopedia

In computer programming, the **strategy pattern** (also known as the **policy pattern**) is a behavioural software design pattern that enables an algorithm's behavior to be selected at runtime. The strategy pattern

- defines a family of algorithms,
- encapsulates each algorithm, and
- makes the algorithms interchangeable within that family.

Strategy lets the algorithm vary independently from clients that use it.[1] Strategy is one of the patterns included in the influential book *Design Patterns* by Gamma et al. that popularized the concept of using patterns to describe software design.

For instance, a class that performs validation on incoming data may use a strategy pattern to select a validation algorithm based on the type of data, the source of the data, user choice, or other discriminating factors. These factors are not known for each case until run-time, and may require radically different validation to be performed. The validation strategies, encapsulated separately from the validating object, may be used by other validating objects in different areas of the system (or even different systems) without code duplication.

The essential requirement in the programming language is the ability to store a reference to some code in a data structure and retrieve it. This can be achieved by mechanisms such as the native function pointer, the first-class function, classes or class instances in object-oriented programming languages, or accessing the language implementation's internal storage of code via reflection.

Strategy Pattern Write-up From: https://sourcemaking.com/design_patterns/strategy

Intent

- Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from the clients that use it.
- Capture the abstraction in an interface, bury implementation details in derived classes.

Problem

One of the dominant strategies of object-oriented design is the "open-closed principle".

SIDENOTE: In object-oriented programming, the **open**/**closed principle** states "software entities (classes, modules, functions, etc.) should be **open** for extension, but **closed** for modification"; that is, such an entity can allow its behavior to be extended without modifying its source code.

Figure 1 demonstrates how this is routinely achieved:

- encapsulate interface details in a base class, and bury implementation details in derived classes.
- Clients can then couple themselves to an interface, and not have to experience the upheaval associated with change: no impact when the number of derived classes changes, and no impact when the implementation of a derived class changes.

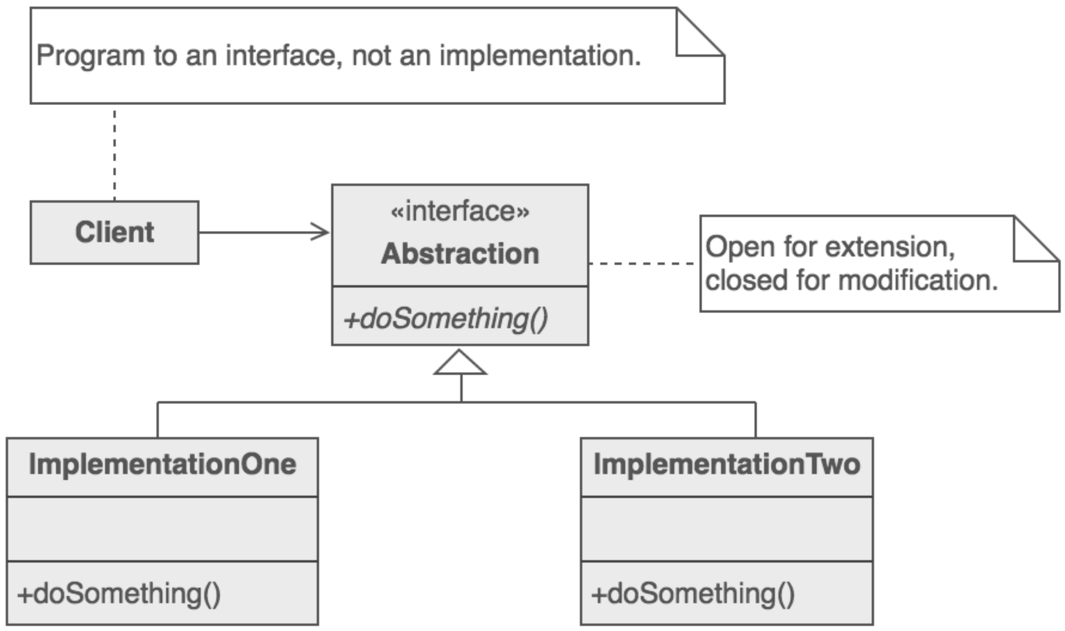


Figure 1 Class Diagram for a Typical Strategy Pattern Implementation

A generic value of the software community for years has been, "maximize cohesion and minimize coupling". The object-oriented design approach shown in Figure 1 is all about minimizing coupling. Since the client is coupled only to an abstraction (i.e. a useful fiction), and not a particular realization of that abstraction, the client could be said to be practicing "abstract coupling" . an object-oriented variant of the more generic exhortation "minimize coupling".

A more popular characterization of this "abstract coupling" principle is "Program to an interface, not an implementation".

Clients should prefer the "additional level of indirection" that an interface (or an abstract base class) affords. The interface captures the abstraction (i.e. the "useful fiction") the client wants to exercise, and the implementations of that interface are effectively hidden.

Structure

The Interface entity could represent either an abstract base class, or the method signature expectations by the client. In the former case, the inheritance hierarchy represents dynamic polymorphism. In the latter case, the Interface entity represents template code in the client and the inheritance hierarchy represents static polymorphism.

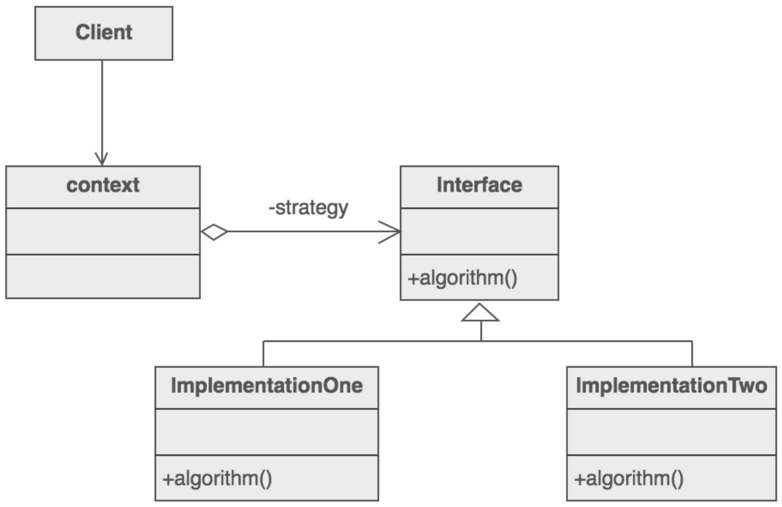


Figure 2 Class Diagram for an Alternate Implementation of the Strategy Pattern

Example

A Strategy defines a set of algorithms that can be used interchangeably.
Modes of transportation to an airport is an example of a Strategy. Several options exist such as driving one's own car, taking a taxi, an airport shuttle, a city bus, or a limousine service. For some airports, subways and helicopters are also available as a mode of transportation to the airport. Any of these modes of transportation will get a traveler to the airport, and they can be used interchangeably. The traveler must choose the Strategy based on trade-offs between cost, convenience, and time.

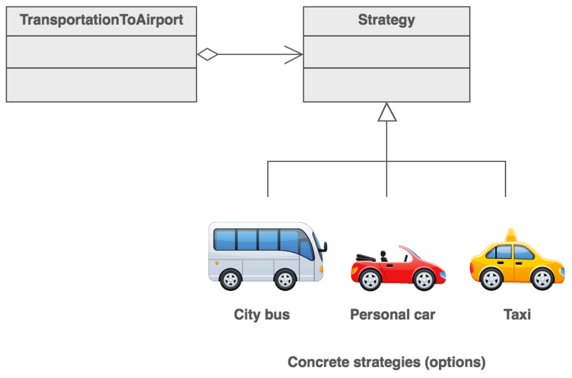
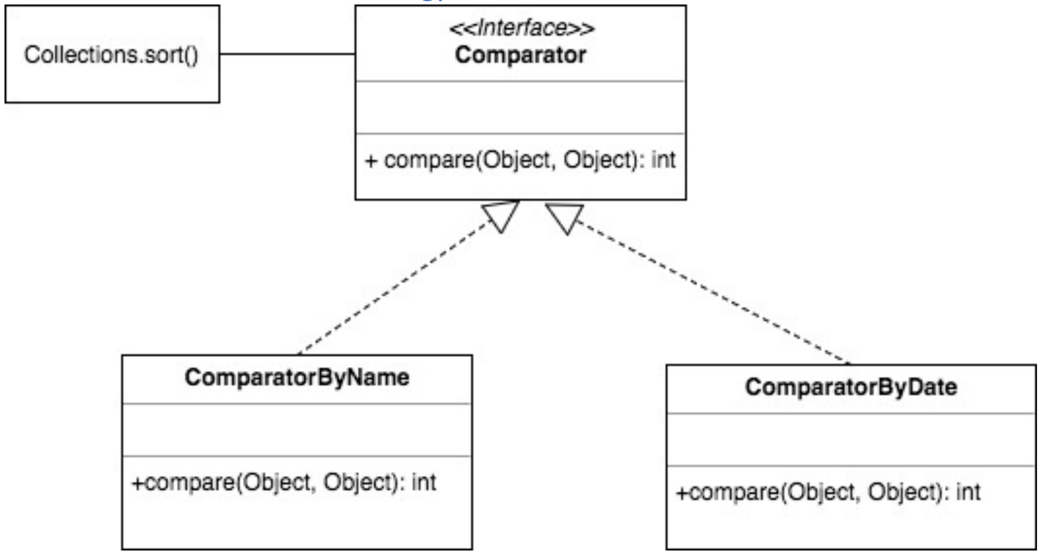


Figure 3 Example of the Strategy Pattern in Action

Check List

1. Identify an algorithm (i.e. a behavior) that the client would prefer to access through a "flex point".
2. Specify the signature for that algorithm in an interface.
3. Bury the alternative implementation details in derived classes.
4. Clients of the algorithm couple themselves to the interface.

Where Have We Seen the Strategy Pattern?


Collections.sort()
<<Interface>>
Comparator
+ compare(Object, Object): int
ComparatorByName
+compare(Object, Object): int
ComparatorByDate
+compare(Object, Object): int

Strategy Pattern Description from Our Textbook

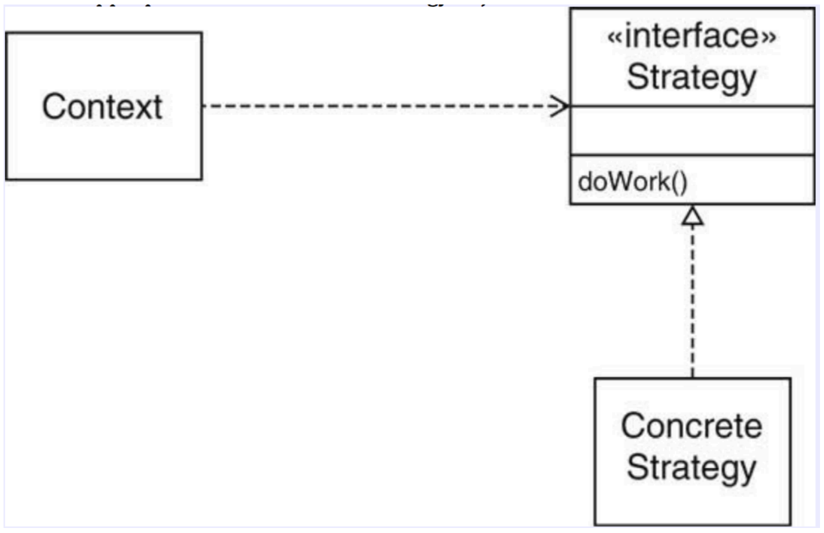
Pattern: Strategy

Context:

1. A class (which we'll call the context class) can benefit from different variants of an algorithm.
2. Clients of the context class sometimes want to supply custom version fo the algorithm.

Solution:

1. Define an interface type that is an abstraction for the algorithm. We'll call this interface type the strategy.
2. Concrete strategy classes implement the strategy interface type. Each strategy class implements a version of the algorithm.
3. The client supplies a concrete strategy object to the context class.
4. Whenever the algorithm needs to be executed, the context class calls the appropriate methods of the strategy object.



In Chapter 4, you encountered a different manifestation of the STRATEGY pattern. Recall how you can pass a `Comparator` object to the `Collections.sort` method to specify how elements should be compared.

```
Comparator comp = new CountryComparatorByName();
Collections.sort(countries, comp);
```

The comparator object encapsulates the comparison algorithm. By varying the comparator, you can sort by different criteria. Here is the mapping from the pattern names to the actual names:

Name in Design Pattern	Actual Name
`Context`	`Collections`
`Strategy`	`Comparator`
`ConcreteStrategy`	A class that implements the `Comparator` interface type
`doWork()`	`compare()`

Best practice for solving problems, gang of four (GOF), 23, common recurring programming challenges, shared language, related, discoverable, NOT, part of API,

Class 17 – Observer Pattern and Model-View-Controller Architecture

Learning Objectives

- Understand the concept of a Model-View-Controller Architecture. Explain the role of the model, the view, and the controller.
- Understand the benefits of the MVC architecture
- Understand the Observer Pattern and consider examples of how it might be applied
- Perform in-class reflective activity.

In-class Reflective Activity

For the upcoming group project, write your reflections on the following two considerations.

- What are you concerned about regarding your ability to contribute to your group?
- What makes you a great group member?

Observer Pattern

The observer pattern is the idea behind the **Model-View-Controller (MVC) Architecture**

- The ________________________________ holds the information in a data structure

  (array, tree, queue, ….). This is just the ____________________. It does not have a visual

  appearance. The model knows nothing of how it ________________________________.

- Other objects, __________________________, draw the visible parts of the data, in a format specific to the view. (For example, you might display data as a graph or bar chart or table).

  The views are interested in ______________________________ of the model. The views don't know about the controllers.

- Each view also has a ____________________________, an object that processes user

  interaction (____, etc).

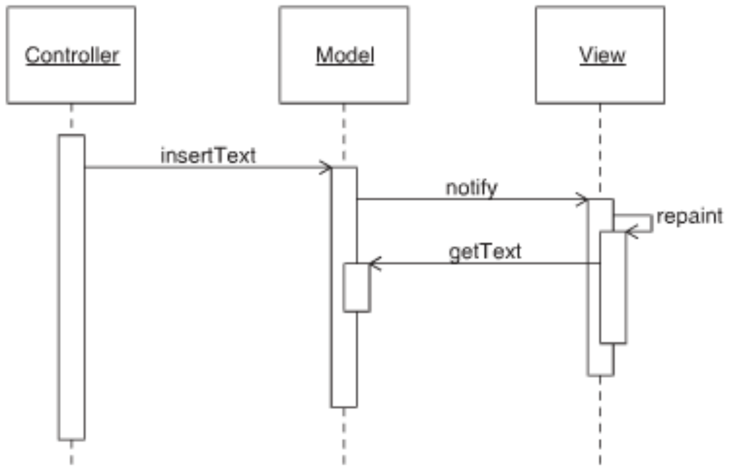
- The model knows about the number of views. Model knows nothing in detail about the observers except that it should notify them whenever the model data changes.

Why is this a great thing?

This makes it easy to
____.

It also makes it easy to support multiple
____.

EX: Suppose you want to add text to your user interface



PATTERN

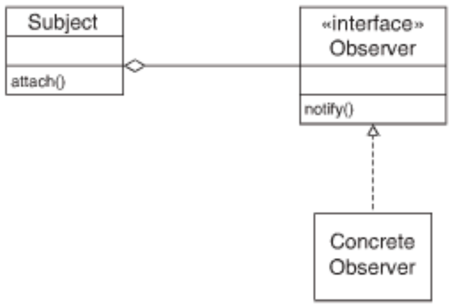
OBSERVER

Context

1. An object (which we'll call the *subject*) is the source of *events* (such as "my data has changed").
2. One or more objects (called the *observers*) want to know when an event occurs.

Solution

1. Define an observer interface type. Observer classes must implement this interface type.
2. The subject maintains a collection of observer objects.
3. The subject class supplies methods for attaching observers.
4. Whenever an event occurs, the subject notifies all observers.



This pattern is used frequently in graphics/user interface design. EX: ActionListener class in the Java API.

Observer Design Pattern & Examples

from http://www.java2blog.com/2013/02/observer-design-pattern-in-java.html

As the name suggests it is used for observing some objects. Observers watch for any change in state or property of subject. Suppose you are interested in particular object and want to get notified when its state changes. Then you observe that object and when any state or property change happens to that object, it notifies you.

As described by GoF:
"Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically".

You can think of observer design pattern in two ways

- **Subject-Observer relationship**: The object, which is being observed, is referred to as the Subject and the classes, which observe the subject, are called Observers.

- **Publisher-Subscriber relationship**: A publisher is one who publishes data and notifies the list of subscribers who have subscribed for the same to that publisher. A simple example is a Newspaper. Whenever a new edition is published by the publisher, it will be circulated among subscribers whom have subscribed to publisher.

The observers will not monitor the object for state since observers will be notified for every state change of subject, until they stop observing subject. So the subject (or publisher) follows the Hollywood principle: "Don't call us, we will call you".

Some real life examples of Observer Pattern

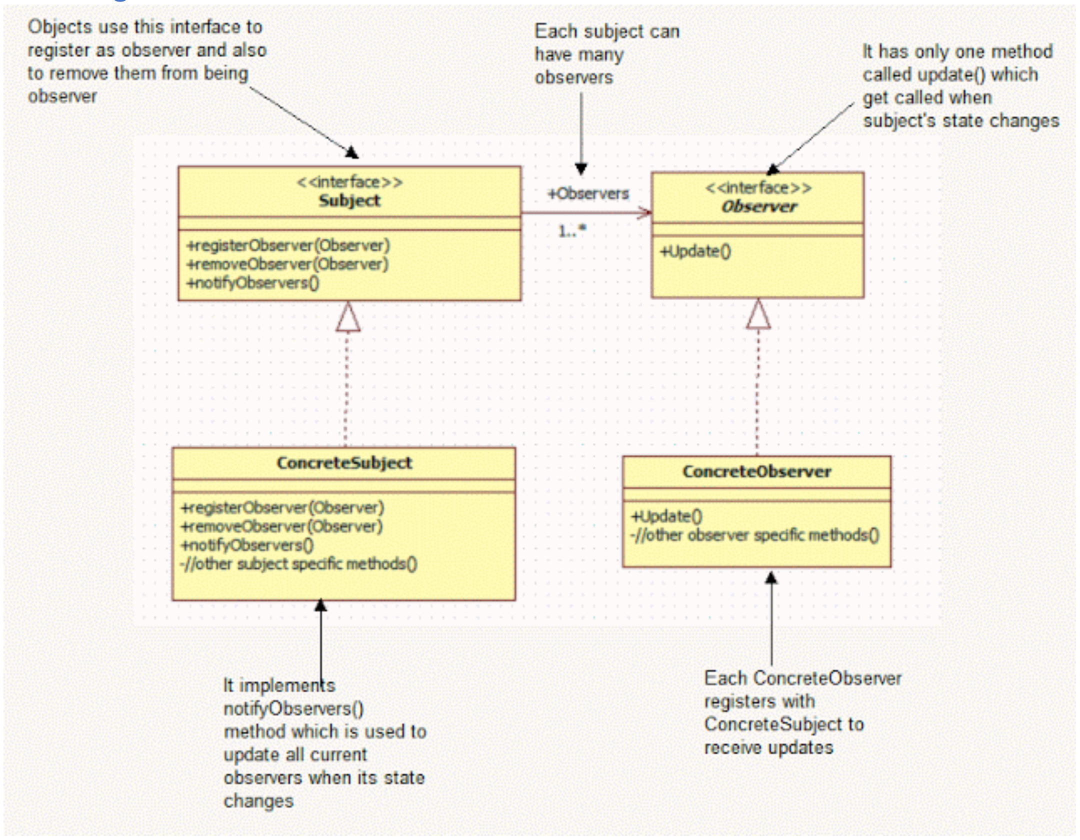
Suppose you're shopping online. Some sites are setup so that when you search for a product and it is unavailable, you see an option to "Notify me when product is available." If you click this option and provide your email address, you subscribe that product. When the state of product changes (i.e. it becomes available), you will get notification email. In this case, the Product is the subject and you are an observer.

Think about Facebook. If you subscribe to someone (by putting them on your friend or close friend list), then whenever they post new updates, you will be notified.

When to use the Observer Pattern:

- When one object changes its state, then all other dependent objects must automatically change their state to maintain consistency.
- When the subject doesn't know about the number of observers it has.
- When an object should be able to notify other objects without knowing who those objects are.

UML Diagram for Observer Pattern


Objects use this interface to register as observer and also to remove them from being observer
Each subject can have many observers
It has only one method called update() which get called when subject's state changes
<<interface>> Subject
+registerObserver(Observer)
+removeObserver(Observer)
+notifyObservers()
+Observers
1..*
<<interface>> Observer
+Update()
ConcreteSubject
+registerObserver(Observer)
+removeObserver(Observer)
+notifyObservers()
-//other subject specific methods()
ConcreteObserver
+Update()
-//other observer specific methods()
It implements notifyObservers() method which is used to update all current observers when its state changes
Each ConcreteObserver registers with ConcreteSubject to receive updates

Components:

Subject

- Knows its observers
- Has any number of observer
- Provides an interface to attach and detaching observer object at run time

Observer

- Provides an update interface to receive signal from subject

ConcreteSubject

- Stores state of interest to ConcreteObserver objects.
- Sends notification to it's observer

ConcreteObserver

- Maintains reference to a ConcreteSubject object
- Maintains observer state consistent with subjects.
- Implements update operation

Model, data and business algorithms, will be deployed, the view, getting updates, controller, menus-dialogs-mouse movement, update the three pieces, views,

Class 18 – Abstract Classes and Template Pattern

The topics today are a little disjoint. They wrap up our coverage of relationships between classes and how to build extendable code

Learning Objectives

- Understand the concept of an abstract class.
- Understand similarities and differences between abstract classes and interface types
- Understand the purpose of the Template pattern. Identify times when you might use this in a program.
- Understand the side effects of making data and/or methods protected rather than private
- Understand the term refactoring

Abstract classes

Abstract classes are a sort of blend between an ______________________ and a ________________________________.

Abstract classes are classes that can contain ____________________________________ like a regular class.

However, unlike a regular class not all of the methods contain an _________________________________.
Some of them are just headers (like the methods in an interface).

RULE: You cannot ____ an abstract class. Why?

You can inherit from an abstract class and when doing so, must ____.

It's also legal to have variables whose type is an abstract class.

Example:
`SelectableShape` is an abstract class. `HouseShape()` is a class that extends the `SelectableShape` class. It's legal to say:

```
      SelectableShape shape = new HouseShape();
      //legal because SelectableShape methods are defined in HouseShape
```

Why would we use an abstract class?

They are convenient placeholders for factoring out
____.

So why bother with interfaces? Why not just have abstract classes?

A class can only extend _________________ abstract class, but it can implement _______________ interface types.

In the API, an interface type and an abstract class are often supplied in pairs.

Template Method Pattern

Idea: Supply an algorithm for _____________________________, provided that the sequence of steps does not

depend on _______________________________

How it works:

a ____________________________ defines a method that calls primitive operations that a ___________________ needs

to supply. Each subclass can supply the primitive operations, as is most appropriate for it. The

template method contains the knowledge of ____ the primitive operations into a more complex operation.

You should use this pattern if:

- You have a set of subclass methods that are almost ________________ in implementation
- You can express the differences between the methods as another method. If you can, then

  move the common code into a ___________________. Call the method for the variant part.

You should use this pattern if:

- You have a set of subclass methods that are almost ________________ in implementation
- You can express the differences between the methods as another method. If you can, then

  move the common code into a ___________________. Call the method for the variant part.

PATTERN

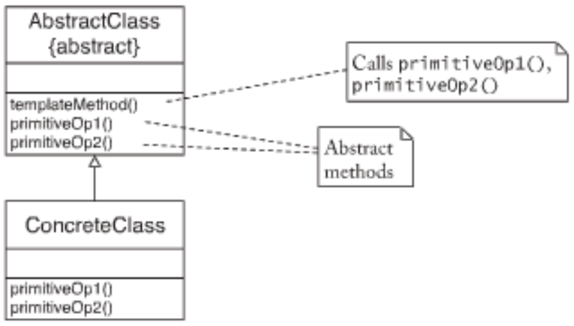
Template Method

Context

1. An algorithm is applicable for multiple types.
2. The algorithm can be broken down into *primitive operations*. The primitive operations can be different for each type.
3. The order of the *primitive operations* in the algorithm doesn't depend on the type.

Solution

1. Define an abstract superclass that has a method for the algorithm and abstract methods for the primitive operations.
2. Implement the algorithm to call the primitive operations in the appropriate order.
3. Do not define the primitive operations in the superclass or define them to have appropriate default behavior.
4. Each subclass defines the primitive operations but not the algorithm.



You should use this pattern if:

- You have a set of subclass methods that are almost ________________ in implementation
- You can express the differences between the methods as another method. If you can, then

  move the common code into a ___________________. Call the method for the variant part.

Remember when we talked about protected?

Our book recommends against using protected methods and data because

It's impossible to predict

How can you be sure that they all need access to the superclass' data and methods?

All classes in a ___________________________also get access to protected data. You might as well make it public.

Recommended:
A class can supply a public interface for all clients and a protected interface for subclasses. Why?

Refactoring

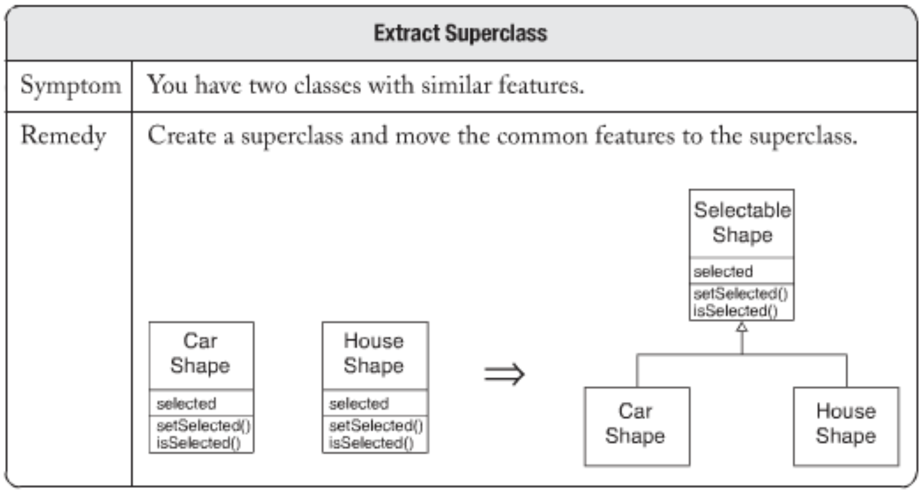
Refactoring means ____________________________ code in a ________________________________ way.

Martin Fowler (author of our UML book) created a list of rules to use when refactoring code.

Obviously, code has to be retested after refactoring to make sure that you haven't broken anything.

Examples of rules for refactoring

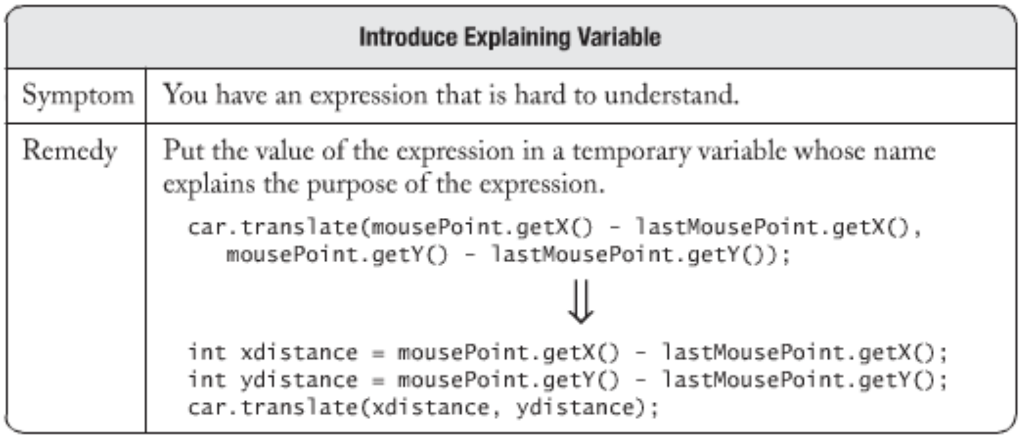
1. Extract superclasses



Extract Superclass	
Symptom	You have two classes with similar features.
Remedy	Create a superclass and move the common features to the superclass.

Car Shape
selected
setSelected()
isSelected()
House Shape
selected
setSelected()
isSelected()
⇒
Selectable Shape
selected
setSelected()
isSelected()
Car Shape
House Shape

1. Introduce Explaining Variables



Introduce Explaining Variable	
Symptom	You have an expression that is hard to understand.

Remedy	Put the value of the expression in a temporary variable whose name explains the purpose of the expression.
	`car.translate(mousePoint.getX() - lastMousePoint.getX(),`
	`   mousePoint.getY() - lastMousePoint.getY());`
	$\Downarrow$
	`int xdistance = mousePoint.getX() - lastMousePoint.getX();`
	`int ydistance = mousePoint.getY() - lastMousePoint.getY();`
	`car.translate(xdistance, ydistance);`

How are the refactoring rules different from the design patterns?

Refactoring tells you____.

Design patterns tell you how to produce
____ (so you hopefully won't need to refactor).

Interface, regular class, data-full methods-method signatures, implementation, instantiate, could call methods that do not exist, implement method signatures without code, similarities, one, multiple/many, multi-step process – create outline of steps that applies to all types, the data implementation, abstract class, subclass, combining, identical, abstract class, subclass

Class 19 – Decorator Pattern (R - 3/30)

Learning Objectives, Assignments, and Readings

- Take Quiz 8.
- Be able to explain the concept of a Decorator Pattern.
- Talk about the Decorator Pattern and consider examples of how it might be applied

The Decorator Pattern

The Decorator Pattern applies wherever a class _________________________________

of another class while ____

In other words:

The key to this pattern is that the decorated component is completely

____

EX:
Suppose there's more info than will fit in your screen; you need a way to maneuver around the screen. How is this handled in your average window?

The scroll bar adds ____ to the underlying

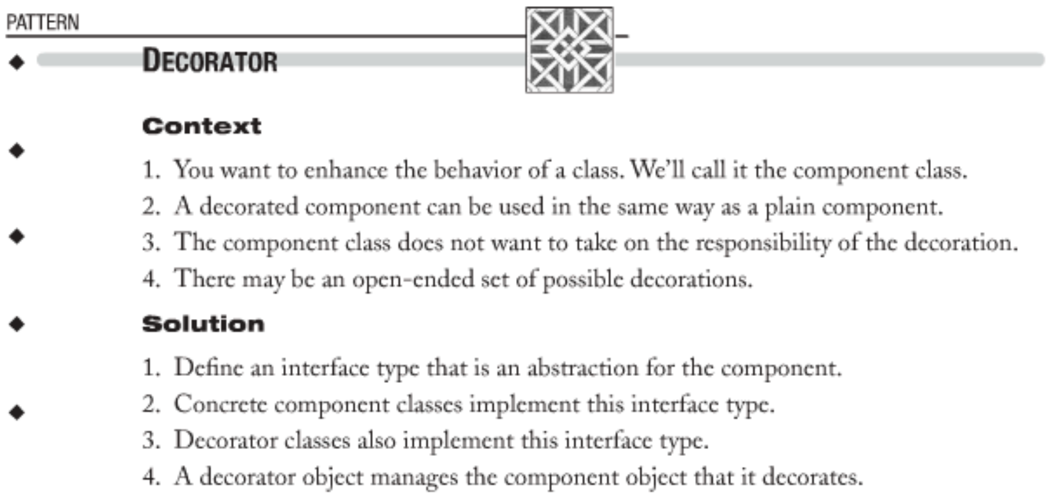
window and is therefore called a ________________________________

Note the window does nothing to acquire a scrollbar. The scrollbar is merely layered onto it.

Advantages:

1. ____
   (it would be a pain of scrollbars had to be written independently for every component - with different variables, options, etc.

2. Your component class can't ____ ways that it may need to be decorated in the future.



PATTERN

DECORATOR

Context

1. You want to enhance the behavior of a class. We'll call it the component class.
2. A decorated component can be used in the same way as a plain component.
3. The component class does not want to take on the responsibility of the decoration.
4. There may be an open-ended set of possible decorations.

Solution

1. Define an interface type that is an abstraction for the component.
2. Concrete component classes implement this interface type.
3. Decorator classes also implement this interface type.
4. A decorator object manages the component object that it decorates.

◆ 5. When implementing a method from the component interface type, the decorator class applies the method to the decorated component and combines the result with the effect of the decoration.

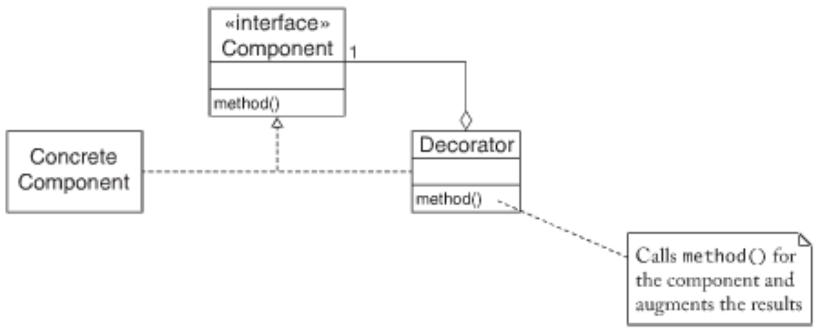
◆

◆

◆

◆

◆



Name in Design Pattern	Actual Name
`Component`	`Reader`
`ConcreteComponent`	`FileReader`
`Decorator`	`BufferedReader`
`method()`	The `read` method. Calling `read` on a buffered reader invokes `read` on the component reader if the buffer is empty.

How to figure out whether a pattern applies to a situation

Go through the ______________________________ and___________________________ parts of the pattern description and make sure that ______________% of the statements apply.

, , Enhance behavior of class without changing original, passive, functionality, decorator, makes code reusable – maintainable, predict, context, solution, 100%

Module 5 – Parallelism

Module 5 Overview

In this module, we study parallelism and concurrency. Each of our class periods have time dedicate to group work. Use you group work time wisely.

Online Parallel Book

http://homes.cs.washington.edu/~djg/teachingMaterials/spac/sophomoricParallelismAndConcurrency.pdf

Parallelism

Described in Classes 21 through 23.

Concurrency

Described in Classes 24 through 26.

Other resources

Video:(Requires login to UMW library to view)
http://umw.kanopystreaming.com.ezproxy.umw.edu/video/parallel-computing-here

Lectures

- C21to23Parallelism.docx
- C24to26Concurrency.docx

Labs

- Lab15Parallelism.docx.

Class 21, 22, and 23 Handout

Classes 21, 22, and 23 used the material in this section.

Learning Objectives, Assignments, and Readings

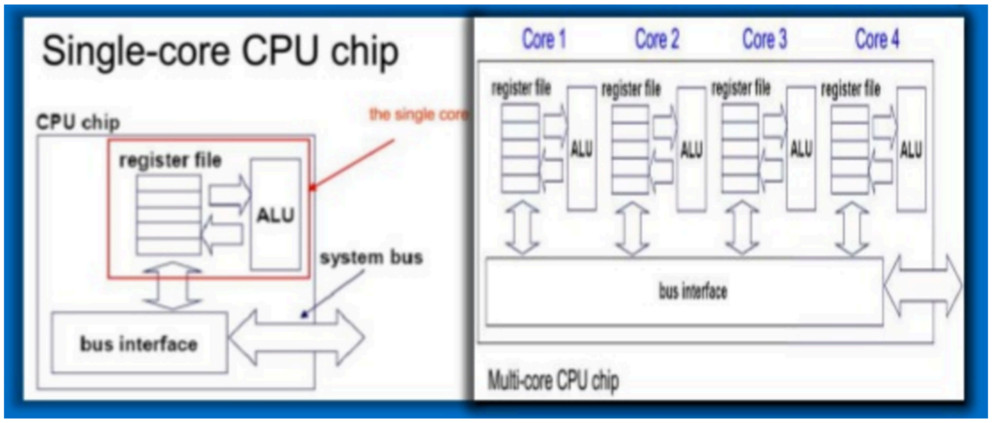
- Half of the class dedicated to the group programming project
- Understand the difference between sequential and parallel processing
- Understand the difference between a parallelism and concurrency
- Learn some Java library methods for simple threading

Sequential Programming vs Parallel Programming

When you run a sequential algorithm, how are statements processed?

When you run a parallel algorithm, how are the statements executed differently than in a sequential algorithm?

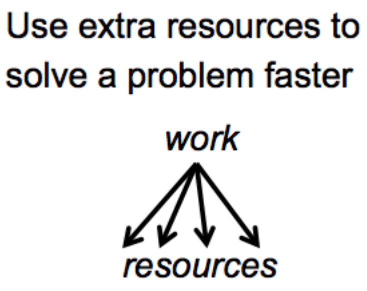
In the computing industry, we're seeing a move away from sequential processing and toward parallelism. Why is there a move toward parallelism?



Parallelism vs Concurrency

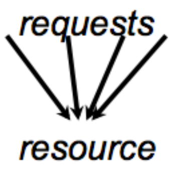
There are no industry standards for these terms yet. The online book that we're using makes a clear distinction between these two very different programming challenges.

____:



____:

Correctly and efficiently manage access to shared resources



There's a subtle difference between parallelism and concurrency. In your own words, restate the cooking related analogy for parallelism. What issues will be important?

Now restate the cooking related analogy for concurrency. What issues will be important?

Focus on Parallelism

Define parallelism:

In all of our discussions, we're going to focus on one approach to parallel algorithms called

________________________________.

We'll implement:

- A set of *threads*, each with its own program counter & call stack
    - No access to another thread's local variables
- Threads can (implicitly) share static fields / objects
    - To *communicate* by write data somewhere another thread reads

There are other possible approaches to parallelism including message-passing, dataflow, and data parallelism. If you get really excited about parallelism, talk to Dr Finlayson. Part of his research involves creating a new programming language that makes parallelism easy even for beginners!

As we create parallel programs, we'll need to be able to do three new things that we don't have to worry about in sequential programs:

1. Have an instruction that allows us to create and run
   ____

   These are called __________________________

2. Have a way to share ________________________

   In Java, we can allow threads to reference the same objects

3. Have an instruction that allows us to _____________________________________

   Think of this as a way to force a thread to wait for another thread to finish

Let's look at the basics provided in the Java API: java.lang.Thread

To get a new thread running:

1. Define a subclass **C** of **java.lang.Thread**, overriding **run**
2. Create an object of class **C**
3. Call that object's **start** method
    - **start** sets off a new thread, using **run** as its "main"

What if we instead called the **run** method of **C**?

Example Problem

Let's look at a simple problem with a simple sequential solution and then think about how to apply parallelism to improve the solution.

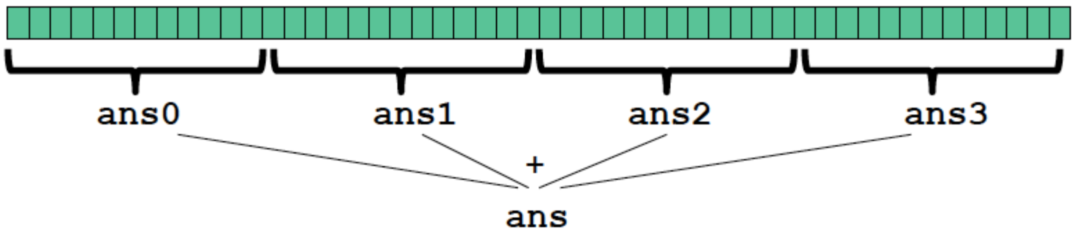
Problem: You have an array of integers and want to calculate the sum of these integers.

Sequential Solution

```
int sum(int[] arr) {
  in tans = 0;
  for (int i = 0; i < arr.length; i++)
    ans += arr[i];
  return ans;
}
```

First stab at a parallel algorithm (assume there are 4 processors):

- Use the first processor to sum the first 1/4 of the array and store the result somewhere.
- Use the second processor to sum the second 1/4 of the array and store the result somewhere.
- Use the third processor to sum the third 1/4 of the array and store the result somewhere.
- Use the fourth processor to sum the fourth 1/4 of the array and store the result somewhere.
- Add the 4 stored results and return that as the answer.



Parallel algorithm in pseudocode:

```
int sum(int[] arr) {
  results = new int[4];
  len = arr.length;
  FORALL(i=0; i < 4; ++i) { // parallel iterations
     results[i] = sumRange(arr,(i*len)/4,((i+1)*len)/4);
  }
  return results[0] + results[1] + results[2] + results[3];
}
int sumRange(int[] arr, int lo, int hi) {
   result = 0;
   for(j=lo; j < hi; ++j)
      result += arr[j];
   return result;
}
```

Draw a 4 item array. Convince yourself that the algorithm works. If the array only contains 4 items, do you think you'll get a speedup over a conventional sequential algorithm?

Note the **FORALL** loop, which is like a **for** loop, but represents the fact that each iteration can be done in parallel. You're promising that all the iterations can be

done ______________________________ without _____________________________with each other.

There is no FORALL statement in Java. Instead, we'll need to:

- Define a subclass **C** of **java.lang.Thread**, overriding **run**
- Create 4 *thread objects* of class C, each given a portion of the work
- Call **start()** on each thread object to actually *run* it in parallel
- *Wait* for threads to finish using **join()**
- Add together their 4 answers for the *final result*

We'll be building a solution on the next few pages. Realize that the first attempts will have errors that we'll correct to learn how to carry out the steps correctly.

Incorrect Attempt #1:
Let's see if we can find all the bulleted steps listed above in the solution. If we can't that's a problem.

```
class SumThread extends java.lang.Thread {
   int lo; // fields for communicating inputs
   int hi;
   int[] arr;
   int ans = 0; // for communicating result
   SumThread(int[] a, int l, int h) {
     lo=l; hi=h; arr=a;
   }

   public void run() { // overriding, must have this type
    for(int i=lo; i<hi; i++)
      ans += arr[i];
   }
}
```

What step does this part of the code implement?

Why can't we pass data into the run( ) method as a parameter?

```
class C {
  static int sum(int[] arr) {
    int len = arr.length;
    int ans = 0;
    SumThread[] ts = new SumThread[4];
    for(int i=0; i < 4; i++) {
      ts[i] = new SumThread(arr,(i*len)/4,((i+1)*len)/4);
    }
    for(int i=0; i < 4; i++) {
      ans += ts[i].ans;
    }
    return ans;
  }

}
```

What steps are shown above?

What steps are missing?

We need to call _______________ on each thread

We need to *Wait* for threads to finish using **join()**

Incorrect Attempt #2:

From here out, we'll abbreviate the SumThread class by hiding the details of the run() method. Everything about it was correct in the first example, so we won't need to make any changes to it.

```
class SumThread extends java.lang.Thread {
   int lo; // fields for communicating inputs
   int hi;
   int[] arr;
   int ans = 0; // for communicating result
   SumThread(int[] a, int l, int h) {
     lo=l; hi=h; arr=a;
   }
   public void run() { // overriding, must have this type
     for(int i=lo; i<hi; i++)
       ans += arr[i];
   }
}
class C {
  static int sum(int[] arr) throws java.lang.InterruptedException {
    int len = arr.length;
    int ans = 0;
    SumThread[] ts = new SumThread[4];
    for(int i=0; i < 4; i++) {
      ts[i] = new SumThread(arr,(i*len)/4,((i+1)*len)/4);
      ts[i].start(); // start not run
    }
    for(int i=0; i < 4; i++) {
      ans += ts[i].ans;
    }
    return ans;
   }
}
```

What steps are shown in the sum( ) method above?

- The sum() method creates 4 instances of the SumThread class
- SumThread is a subclass of java.lang.Thread
- Notice the instruction in red. We call start() on each instance of SumThread which will create our parallel threads.
- Each instance gets a unique subset of the array indices to work on

(note: low bound is included and high bound is excluded for each SumThread so there's no overlap)

- Each SumThread object has an **ans** field. This is a shared memory location for communicating the thread's answer back to the main thread.
- Then the main thread sums the 4 ans fields to calculate the final sum
- But the problem is ____. The main thread doesn't wait for the helper threads to complete before calculating the sum.

Solution: we need to delay the second for loop. We don't want it to execute until all of the helper threads have terminated.

Attempt 3: Correct in spirit but not the best solution.

We need a new instruction to tell the computer to wait for threads to finish. The new instruction is called join().

```
class SumThread extends java.lang.Thread {
   int lo; // fields for communicating inputs
   int hi;
   int[] arr;
   int ans = 0; // for communicating result
   SumThread(int[] a, int l, int h) {
     lo=l; hi=h; arr=a;
   }
   public void run() { // overriding, must have this type
     for(int i=lo; i<hi; i++)
       ans += arr[i];
   }
}
class C {
  static int sum(int[] arr) throws java.lang.InterruptedException {
    int len = arr.length;
    int ans = 0;
    SumThread[] ts = new SumThread[4];
    for(int i=0; i < 4; i++) {
      ts[i] = new SumThread(arr,(i*len)/4,((i+1)*len)/4);
      ts[i].start(); // start not run
    }
    for(int i=0; i < 4; i++) {
      ans += ts[i].ans;
      ts[i].joint(); // wait for helper to finish
    }
    return ans;
  }
}
```

Note the new line in red.

Now the main thread will block until the first thread terminates. Then until the second thread terminates, etc.

Note that to create the equivalent of the FORALL statement, we've used 2 loops. The first creates all of the threads and starts them. Then the second loop blocks and waits for them all to finish.

This is called ____ parallelism.

Describe what each of the following methods in the java.lang.Thread library does:

Start()

Run()

Join()

Building a Better Algorithm

But we said the previous algorithm was "correct in spirit", so what's wrong with it? It gives us the correct answer. It uses parallelism.

Can you think of any limitations to our solution?

Hint: What if our computer had 8 cores? What if our computer only had 3 cores available? How would this affect the execution of the solution that we wrote?

Big picture issues:

1. We should ____________________________ the number of threads.

```
static int sum(int[] arr, int numThreads) throws
java.lang.InterruptedException {
  int len = arr.length;
  int ans = 0;
  SumThread[] ts = new SumThread[numThreads];
  for(int i=0; i < numThreads; i++) {
    ts[i]=new SumThread(arr,(i*len)/numThreads,((i+1)*len)/numThreads);
    ts[i].start();
  }
  for(int i=0; i < numThreads; i++) {
    ts[i].join();
    ans += ts[i].ans;
  }
  return ans
}
```

Summarize the differences between this version and the “Attempt #3 Correct in Spirit” version.

2. We should use only the processors ________________________________

Some processors might be used by other programs.

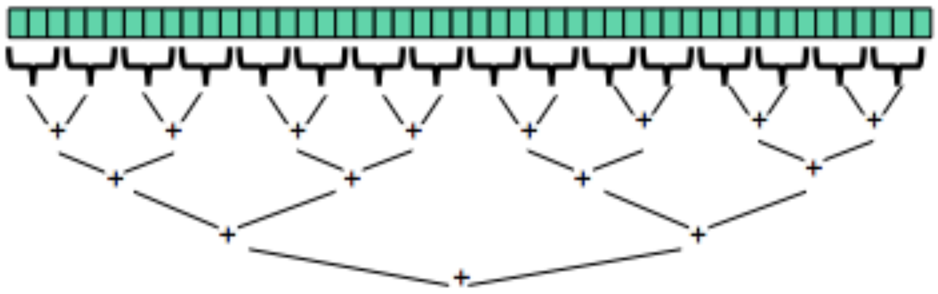
3. It’s possible that the chunks divvied up to the various processors take ________________________ amounts of time.

What’s meant by a load imbalance?

The Best Approach

We're going to need to change our algorithm a bit.

Describe the divide and conquer strategy for the problem (the counterintuitive strategy to subdivide the problem into tiny chunks.)



Here's the code:

```
class SumThread extends java.lang.Thread {
  int lo; // fields for communicating inputs
  int hi;
  int[] arr;
  int ans = 0; // for communicating result
  SumThread(int[] a, int l, int h) { arr=a; lo=l; hi=h; }
  public void run() {
    if (hi - lo == 1) {
     ans = arr[lo];
    } else {
      SumThread left  = new SumThread(arr,lo,(hi+lo)/2);
      SumThread right = new SumThread(arr,(hi+lo)/2,hi);
      left.start();
      right.start();
      left.join();
      right.join();
      ans = left.ans + right.ans;

    }
  }
}

int sum(int[] arr) {
   SumThread t = new SumThread(arr,0,arr.length);
   t.run();
   return t.ans;
}
```

But how to we pick the right chunk size for each processor? If we pick chunks of size 1 (each processor processes 1 element), then this solution's final step to combine the sums is identical to the sequential algorithm. That's not a speedup at all!

We've compensated for this by assigning a CUTOFF above. If the chunk size is less than this value, it uses a sequential algorithm. For larger chunks, it uses the parallel algorithm.

This will give the correct solution but….. creating all those threads isn't free. It takes time. There's a lot of _________________________ involved in creating Java threads in this way. We'll need a different library to maximize our speedup.

Another reason to use a different library is that the java.lang.Threads library requires you to do a lot of the optimization by hand. That's a lot of work and it's hard to get the cutoff selected correctly.

There's a ForkJoin Framework specifically for divide and conquer parallel algorithms. This is going to solve all our problems.

Don't subclass `Thread`	Do subclass `RecursiveTask<V>`
Don't override `run`	Do override `compute`
Do not use an `ans` field	Do return a `V` from `compute`
Don't call `start`	Do call `fork`
Don't just call `join`	Do call `join` which returns answer
Don't call `run` to hand-optimize	Do call `compute` to hand-optimize
Don't have a topmost call to `run`	Do create a pool and call `invoke`

Final code using fork-join framework

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;
class SumArray extends RecursiveTask<Integer> {
   static int SEQUENTIAL_THRESHOLD = 1000;
   int lo;
   int hi;
   int[] arr;
   SumArray(int[] a, int l, int h) { lo=l; hi=h; arr=a; }
   public Integer compute() {
      if(hi - lo <= SEQUENTIAL_THRESHOLD) {
         int ans = 0;
         for(int i=lo; i < hi; ++i)
            ans += arr[i];
         return ans;
      } else {
          SumArray left  = new SumArray(arr,lo,(hi+lo)/2);
          SumArray right = new SumArray(arr,(hi+lo)/2,hi);
          left.fork();
          int rightAns = right.compute();
          int leftAns = left.join();
          return leftAns + rightAns;
      }
   }
}

class Main {
  static int sumArray(int[] array) {
    return ForkJoinPool.commonPool().invoke(new
SumArray(array,0,array.length));

//Alternative
     static final ForkJoinPool fjPool = new ForkJoinPool();
     return fjPool.invoke(new SumArry(array,0,array.length)
  }
}
```

Class 24, 25, and 26 Handout

Classes 24, 25, and 26 used the material in this section.

Learning Objectives, Assignments, and Readings

- Understand common challenges with concurrent programs
- Understand how mutual exclusion prevents bad interleaving

- List and describe the 3 operations included in a lock
- Know the 3 ways that data can be protected within a program
- Understand the tradeoffs between performance and critical sections
- Define the term deadlock and describe how it can occur

Challenge of concurrency

The challenge of concurrency is to control access by multiple threads to a shared resource (or resources).

ATM example

Two people share a checking account. It has a balance of $100.

At 9:00, person 1 logs in and checks the balance. At 9:01, person 1 requests to withdraw $100. This is approved at 9:03 & person 1 has the money in hand.

At 9:00, person 2 logs in and checks the balance. It's $100. At 9:02, person 2 requests to withdraw $100. The bank approves and at 9:03, person 2 has the money in hand.

Is this operation good business?

Trinkle Vending Machine example:

Not writing this one down to protect my information sources. Trust me, you'll remember it.

Summary of issues:

The typical model we'll consider has multiple threads predominantly operating

___________________________________. For example, perhaps we create a different thread to handle each customer's bank requests.

Typically, these threads won't need to access the same account and can proceed

without ________________________________ and without ______________________________. Occasionally, the threads will access the same account and will need to coordinate before performing particular activities.

Warning: These types of programs are really hard to debug. You (the programmer) can't necessarily control the order in which events occur. It's hard (sometimes) to reproduce buggy behavior. And if you can't reproduce it, that makes it hard to find problems.

Understanding Issues

Consider the bank example. Here's a typical sequential program to handle the typical bank operations.

```
class BankAccount {
      private int balance = 0;

      int getBalance() { return balance; }
      void setBalance(int x)  {balance = x; }
      void withdraw(int amount) {
            int b = getBalance();
            if (amount > b)
                  throw new  WithdrawTooLargeException();
            setBalance(b – amount);
      }
      //other operations like deposit, …
}
```

If two threads are operating and one calls

```
x.withdraw(100);
```

and the other calls

```
y.withdraw(100);
```

Will it cause a bank error if….

- x & y are different objects referring to distinct bank accounts?

- x & y refer to the same account, but one request is guaranteed to finish it's call before the other call begins?

- x & y refer to the same account and the two calls happen at the same time?

In this last case, we say the calls ______________________________________.
Interleaving calls can cause problems.

Example of Interleaving Issue

```
Thread 1                              Thread 2
---------------------                 -----------------------
int b = getBalance();

                                      int b = getBalance();
                                      if (amount > b)
                                            throw new …;
                                      setBalance(b-amount);

if (amount > b)
      throw new …;
setBalance(b – amount);
```

What’s the result?

Is this what should have happened?

Using Mutual Exclusion to Prevent Bad Interleaving

To prevent interleaving, we need to use a technique called ____________________________.

Mutual exclusion means we’ll only allow one thread to access a particular chunk of code at a time.

This allows us to create a ____ : a sequence of operations for which interleaving will not be allowed.

To use mutual exclusion we have to use synchronization primitives defined in the programming languages. This is something that you can _______________________ write from scratch.

Primitives for Creating a Critical Section

We’re going to use a primitive called a _____________________________________
It’s an Abstract Data Type (ADT) with 3 operations:

____________________ : creates a new lock that is “not held” initially

___________________ : takes a lock and blocks until it is currently “not held” (which may be immediately). It sets the lock to “held” and returns.

__________________ : takes a lock and sets it to “not held”

The lock works like a “do not disturb” sign. We can call the acquire( ) method to turn on the “do not disturb()” sign. The acquire operations doesn’t return until the caller is the thread that most recently hung the sign.

The lock ADT will always make sure that the data structure "does the right thing" no matter how many different threads simultaneously perform acquire operations and/or releases. For example, if 4 threads simultaneously execute an acquire()

operation for a "not held" lock, one will _______________ and return immediately while the other three block. When the winner calls "release()", …

In pseudocode, this might look like:

```
class BankAccount {
      private int balance = 0;
      private Lock lk = new Lock();
      …

      void withdraw(int amount) {
            lk.acquire(); /*may block*/
            int b = getBalance();
            if (amount > b)
                  throw new WithdrawTooLargeException();
            setBalance(b-amount);
            lk.release();
      }
      //deposit would also acquire/release the lock lk
}
```

Now different threads can operate on different accounts at the same time.

And only one thread can operate on a specific account at a specific time (thanks to the lock).

We still have 2 problems in the Code

1. What happens when an exception is thrown?   That thread never

________________________________ the lock. So no other thread can ever have access to that account. This isn't good.

2. We need re-entrant locks. In other words, if a particular thread holds the lock at a particular time and needs to execute code that acquires the lock, the computer should be able to figure out tat this same thread already holds the lock and continue on rather that stopping the thread in it's tracks.

Additional details about how our 3 lock methods work:

`new()` creates a new lock with no current holder and a count of _____________________

`acquire( )` blocks if there is a current holder
____.
Otherwise if the current holder is the thread calling it, don't block and increment the counter.
Otherwise there is no current holder, so set the current holder to the calling thread.

`release()` only releases the lock (sets the current holder to "none") if the count is 0. Otherwise it ______________________________

How this works in Java syntax

```
	synchronized (expression) {
		statements
	}
```

This looks very like a while loop, but it's not a loop.

How it works:

1. The expression is evaluated. It must produce (a reference to) an object – not null or a number. This object is treated as a lock. In Java, every object is a lock that any thread can acquire or release. This decision is a little weird, but it's convenient in an object-oriented language.

2. The synchronized statement acquires the lock. ie The object that is the result of step 1. This may block until the lock is available. Locks are re-entrant in Java, so the statement will not block if the executing thread already holds it.

3. After the lock is successfully acquired, the statements are executed.

4. When control leaves the statements, the lock is released. This happens either when the final } is reached OR when the program "jumps out of the statement" via an exception, a return, a break, or a continue statement.

Note that the lock is released at the ending }
This is true even if an exception only causes a part of the "body" not to complete

EX:
```
	class BankAccount {
		private int balance = 0;
		private Object lk = new Object();
		int getBalance() {
			synchronized (lk) {
				return balance;
			}
		}
		void setBalance(int x) {
			synchronized (lk) {
```

```
                balance = x;
            }
        }
        void withdraw(int amount) {
            synchronized (lk) {
                int b = getBalance();
                if (amount > b)
                    throw new WithdrawTooLargeException();
                setBalance(b-amount);
            }
        }
        //deposit and other operations would also use synchronized(lk)
    }
```

Or we can save a step and NOT make a new object just for the lock. Instead we can use the calling object like this

EX:

```
    class BankAccount {
        private int balance = 0;
        int getBalance() {
            synchronized (this) {
                return balance;
            }
        }
        void setBalance(int x) {
            synchronized (this) {
                balance = x;
            }
        }
        void withdraw(int amount) {
            synchronized (this) {
                int b = getBalance();
                if (amount > b)
                    throw new WithdrawTooLargeException();
                setBalance(b-amount);
            }
        }
        //deposit and other operations would also use synchronized(lk)
    }
```

And finally, when we want to lock the entire body of a method (as shown above), Java includes a syntax shortcut to reduce typing. The final result would look like this:

EX:

```
    class BankAccount {
        private int balance = 0;
        synchronized int getBalance() {
            return balance;
        }
        synchronized void setBalance(int x) {
            balance = x;
        }
        synchronized void withdraw(int amount) {
              int b = getBalance();
              if (amount > b)
                    throw new WithdrawTooLargeException();
              setBalance(b-amount);
        }
        //deposit and operations would also use synchronized(lk)
  }
```

Data Race

Data race – _______________________________________

Example: one thread might read an object field at the same moment that another thread writes to the same field

Example: one thread might write an object field at the same moment that another thread reads the same object field

It's never an error if two threads just read an object field simultaneously.   Why?

Data Race Rules

Every memory location should either be

1. Thread-local:

2. Immutable:

3. Synchronized:

Bottom line: Avoid sharing objects unless those objects are specifically being used to enable shared-memory communication among threads.

Guidelines

- **Avoid Data Races**: Use locks to ensure that two threads never simultaneously

  ____.

- **Use consistent locking:** For each location that needs synchronization, identify a lock that is always held when accessing that location. This will prevent data races, but not necessarily bad interleavings.

- **Start with coarse-grained locking** and move to finer-grained locking only if

____. Ie. use fewer locks to guard more objects (use one lock to guard an array of accounts). Only move to a separate lock for every single bank account if performance improvements are needed.

- **Make critical sections large enough for correctness but** _________________________. Don't perform I/O or expensive computations w/in critical sections.

- **Think in terms of what operations need to be atomic (_________________________).** Determine a locking strategy after you know what these critical sections are.

- **Don't implement your own concurrent data structures.** Use the carefully tuned ones written by experts and provided in standard libraries.

- **Avoid deadlock.**

Deadlock

What is deadlock? If a collection of threads are blocked forever, all waiting for

another thread in the collection to do something (_____________________________________) then we say the threads are deadlocked.

Deadlock Example

```
Class BankAccount {
      …
      synchronized void withdraw(int amt) {… }
      synchronized void deposit (int amt) {…{
      synchronized void transferFo(int amt, BankAccount a) {
            this.withdraw(amt);
            a.deposit(amt);
      }
}
```

Looks good, right?

- There are no data races because only methods that directly access fields are synchronized– like withdraw and deposit - and these acquire the correct lock.

- transferTo seems atomic. For another thread to see the intermediate state where the withdrawal has occurred but not the deposit, would require operating on the withdraw n-from account. And since transferTo is synchronized, this can't happen.

But this deadlock situation can happen if we're unlucky in the ordering.

EX: One thread is transferring from account A to B, while the other is transferring from B to A

Thread 1: x.transferTo(1, y)	Thread 2: y.transferTo(1.x)
acquire lock from x	
withdraw 1 from x	
	acquire lock from y
	withdraw 1 from y
	block on lock for x
block on lock for y	

The first thread holds the lock for A and is blocked on the lock for B. The second thread holds the lock for B and is blocked on the lock for A. So both are waiting for a lock to be released by a thread that is blocked.

________________________________.

Deadlock Solutions

Deadlock Solution Option 1

Option 1: Don't synchronize the transferTo() method. Because withdraw and deposit will remain synchronized the only downside is that someone might "see" the state where the withdraw has occurred but not the deposit. That might be okay.

Deadlock Solution Option 1

Option 2: Use a coarser grained locking. Ie one lock for __________________________

In both options you avoid deadlock because we've ensured that no thread ever holds

______________________________________.

If it's essential that the critical section acquire two locks, then make a required order that all threads have to use. Ie. If each thread HAS to acquire lock A before it acquires lock B. Then we can never have deadlock.

Labs Overview

Labs are an important component of CPSC 240. Labs allow you to practice programming concepts, explore areas discussed in class, and answer questions to demonstrate your knowledge.

Labs have a regular rhythm. For most of the semester, we will begin a lab each Thursday, and it will be due the following Tuesday.

There will be a few weeks where we will perform a lab on Tuesday.

Lab Computers

I recommend you use your laptop for performing the in-class portion of labs. This simplifies the continuing labs after leaving class. The computers in B12 are available for in-class labs.

Lab Submissions

For each lab requiring a Canvas submission, you must submit a Lab Submission Form and the individual submission requirements stated in the lab description.

Lab Submission Form

Select one of the following.	
	I understood all lab concepts.
	I understood most lab concepts.
	I understood some lab concepts.
	I did not understand lab concepts.
Select one of the following.	
	I completed 100% of the lab.
	I completed 80% of the lab.
	I completed 60% of the lab.
	I completed less than 50% of the lab.

Lab 1 – Java API & ArrayLists

Lab 1 consists of questions and some programming.

Learning Objectives

- To become familiar with the *Java API*
- To explore the incredibly useful *ArrayList* class in the *Java API*
- To *edit*, *compile*, and *run* a *Java program*

NOTE: Throughout this lab, there are some computer science specific terms that are italicized. These are terms that you should be familiar with from previous courses. If you're not, do a little extra Google searching or revisit a book from a previous course to learn about these concepts.

Introduction to the Java API

The *Java API* is the standard *library* of *classes* used in the Java programming language. One of *Java's* strengths is its vast and well-organized *API*. In this course, you can use any classes that are part of the *API* to complete any assignment unless otherwise specified. To become an efficient programmer, you should familiarize yourself with the contents of the *API*. This knowledge allows you to reuse existing classes and not have to write them yourself. Familiarity with the *API* saves time by not having to create code for common data structures and operations.

Throughout this lab, questions are posed. Answer the questions in a Word document or on a piece of paper. You must post your answers on Canvas so if you hand-write answers, you will have to post an image file of photo.

Question 1: What does the acronym *API* stand for?

Learning about the ArrayList class

Open the *Java API* and find the information about the *ArrayList* class.
(You can find this by Googling "*Java API ArrayList*".) All pages in the official *Java API* will start with the URL: http://docs.oracle.com/ . Use the *API* to answer the following questions.

Question 2: The *API* indicates that the *ArrayList* class *extends* the AbstractList class. What does the reserved word "*extends*" indicate?

Question 3: The *API* indicates that the *ArrayList* class *implements* 4 classes: *List*, RandomAccess, Cloneable, and Serializable. What does the reserved word "*implements*" signify about the class relationships here?

Question 4: In CPSC 110 or 220, you probably learned about "*regular*" *arrays* (in Java or another language)? Read through the *API's* description of the *ArrayList* class and identify some ways in which an *ArrayList* is different than a regular array.

Question 5: What does this sentence mean? "This *class* is a member of the *Java* Collections Framework."

Question 6: How many *constructors* does the *ArrayList* class include? Why would a class include more than one *constructor*? You may remember that a *constructor* is a special purpose *method* that sets the initial state of an *object* when it's instantiated. You might remember that having multiple constructors is an example of *function overloading*.

Question 7: Describe at least 3 ways to place a new value into an *ArrayList*. What method would you *call* and what *parameters* are required for each *call*?

Question 8: What is the name of the *method* that checks to see if an *object* exists in an *ArrayList*?

Write a Java Program
Now write a *main* program using the ArrayList class. Be sure to meet the following *requirements*:

- Create an *ArrayList* object
- Store at least 10 values into *ArrayList*. For example, you could store information about CPSC courses offered this semester (CPSC110, CPSC184, CPSC220, CPSC225, CPSC240, CPSC305, CPSC326, CPCS340, CPSC345, CPSC350, CPSC414, CPC419, CPSC430, CPSC470)

Question 9: Are the values stored in some sort of *sorted* order or *unsorted* order by the ArrayList? Suppose you wanted to change this. How could you go about doing that?

Continue to write your program by including statements to do the following:
- Determine how many values are stored in your ArrayList and print that number to *standard output*.
- Ask the user to enter a value (ex: CPSC110 or CPSC999) from *standard input* and then determine whether that value is in the *ArrayList*. Print a message to the user to indicate that the value is or is not found.

What to Submit
Submit the following items as the deliverables for the Lab 1 assignment on Canvas:
- Answers to the 9 questions as text, a .docx, a .pdf, or a .jpeg file
- The .java file for your program

Lab 2 – Java Classes and JavaDoc Comments

Learning Objectives
- To review terminology related to Classes and Objects
- To practice writing a Class to represent an abstract data type
- To practice testing a Class
- To gain experience in creating Javadoc comments and generating the associated HTML

Tutorial – Review of Classes & Objects
The Java folks at Oracle provide nice tutorials about Java related topics. Today, we examine three tutorials on classes and objects at the following link.

http://docs.oracle.com/javase/tutorial/java/javaOO/index.html

Work through the three sections on *Classes*, *Objects* and *More on Classes*. For this lab, you do not need to work through the sections on *Nested Classes* or *Enum Types*.

Questions

Most of this information should be review. Answer the following questions as you read through the tutorial.

1. What are the required components of a class declaration?
2. There are three different types of variables that can be used inside classes: fields, local variables, and parameters. Give an example of data that you might store in each.
3. What does the term "method signature" mean?
4. What does it mean for a method to be "overloaded"? Why might you want to overload a method in Java?
5. What behavior changes when passing a primitive object vs. a reference object as an argument to a method? Which is passed by value and which is passed by reference?
6. True or false? Every class has at least one constructor, even if you don't explicitly include a constructor in your class declaration.
7. How does the Java interpreter determine that an object can be reclaimed by the garbage collector?
8. What does the keyword "this" refer to? Why is it necessary to use the "this" keyword in the non-default constructor for the Point class (in the tutorial example)?
9. What rule of thumb does the tutorial recommend for assigning access levels for data members?

Programming Practice

Create a Java project that includes the following:

- A `BikePart` class that stores information about a bicycle part. The class should store information about the part name, part number, list price, sales price, and whether the part is on sale. Include appropriate methods to manage this data.
- A main program that tests your bicycle part class to ensure it's working properly. Your main program shall read a sequence of lines from the terminal, where the first line is the number of lines containing bicycle part information. Each bicycle part description line shall have the following format.

  ```
  partName,partNumber,price,salesPrice,onSale
  ```

- The following is an example sequence of you typing in room description lines.

  ```
  4
  WTB_saddle,1234567890,33.00,25.58,false
  26inTube,1234567891,7.00,5.58,true
  seatPost,1234567892,17.00,15.21,true
  carbonHandleBars42cm,1234567893,47.00,5.58,false
  ```

- You are to read the lines into a `BikePart` array (e.g., `BikePart[] bpArray`).
- After reading the lines into your `BikePart` array, you are to print all parts that cost less than $20.00. The following is example output from the above input.

  ```
  26inTube,1234567891,7.00,5.58,true
  ```

```
seatPost,1234567892,17.00,15.21,true
```

10. Think about how you know that you have fully tested your class. Turn in output from your program demonstrating that your room class operates correctly. Explain how you know that you've fully tested the code.

JavaDoc Comments

Read this tutorial on Javadoc comments:

https://students.cs.byu.edu/~cs240ta/fall2016/tutorials/javadoctutorial.php

Add Javadoc comments to your `BikePart` class that you created in the Programming Practice section. Include class header JavaDoc and method level JavaDoc.

Use the following tags where appropriate: `@param`, `@return`, `@throws`, and `@author` where appropriate.

Attempt to generate the .html documentation for the Javadoc that you've created. You will need to google for steps on completing this in your IDE of choice. Most of them have a menu option that facilitates the generation of the .html files.

- If you successfully generated HTML JavaDoc, click on the index.html file to open the documentation in a browser. Then take a screenshot and upload it here.
- If you did not successfully generate HTML JavaDoc, include a message indicating unsuccessful and your IDE. We can troubleshoot issues in our next lab.

What to Submit

- Answers to the 10 questions above. You can upload a picture of handwritten answers or an electronic document (.docx, .txt or .pdf).
- A zip file of you .java files for your class and tester.
- A screenshot of your Javadoc documentation or the error message you receive.

Lab 3 – Use Cases, File I/O, and Exceptions

Learning Objectives

- To practice writing use case diagrams.
- To practice writing use cases.
- To select a UML tool.
- To practice writing Java code that performs file input and output.
- To practice writing Java code that processes exceptions.

References

http://agile.csc.ncsu.edu/SEMaterials/UMLOverview.pdf

Part 1: Writing Use Cases

The section at the end of this lab shows a sample use case diagram for a system that has three actors – Cashier, Stocker, and Manager – where each performs actions on the system.

Project 3 has several use case scenarios for four actors - System Administrator, Office Manager, Warehouse Manager, and Sales Associate. Read the Project 3 use case scenarios.

- Create a use case diagram for the project 3 use case scenarios. The goal of a use case diagram is to show the actors (usually various classifications of users) and the functions that they will perform with the system. The actors are shown as labeled stick figures. The functions are shown as labeled ovals. You draw a line to connect an actor to a function. A sample use case diagram is included in the last section of this lab.
- Select two of the use case scenarios and create the use case write up. Be sure to include a title, description and main flow for each. Include an alternate flow where appropriate. If there is no possible alternate flow of events, indicate that as well.

Selecting a UML Tool

- You can use any tool that you'd like to create the diagrams; however, I recommend https://www.draw.io. Draw.io is a web-based tool that you can use for free. Draw.io supports UML and other drawing symbol packages. Draw.io allows you to save your drawing a .xml file on your computer, which can be reopened in Draw.io. Draw.io also allows you to save your drawing as a PDF file on your computer, which can be viewed, printed, and submitted on Canvas.
- We have Microsoft Visio and Rational Rose installed in B12. Both tools are commonly used in industry and would be nice skills to add to your resume. However, they are more complex than Draw.io, and I am not familiar with them.
- Gliffy is another web-based tool – http://www.gliffy.com/uses/uml-software

Part 2: Using File I/O in Java

The goal is to feel comfortable reading and writing text files in Java, by studying some examples. For reading text files, I usually connect a `Scanner` to a `File`, which I find rather intuitive; however, I find writing text files in Java difficult to understand because there are so many options. As you go through these examples, you can determine which style best suits you. Later we will learn how to read and write a Serialized Object files, which can be helpful in our group project.

File I/O Tutorials

Read one or more of these tutorials about using File I/O in Java. These are some good choices, but you're also welcome to find your own.

- https://docs.oracle.com/javase/tutorial/essential/io/file.html
- http://www.tutorialspoint.com/java/java_files_io.htm
- https://www.caveofprogramming.com/java/java-file-reading-and-writing-files-in-java.html

I recommend starting with the Java text file I/O technique that best suits your style. Then move on to the program in the next subsection. If you get confused and cannot find a solution in the tutorial you originally selected, try one of the other sources. The following are two methods performing file I/O[4]. You will notice that file I/O involves `IOExceptions`, which methods must either catch or throw. The examples below catch the exceptions, terminating the program if they occur. In cases where your program prompts a user to input a filename, your program should `catch` the `FileNotFoundException` and inform the user to enter another filename, which of course is a loop because users can repeatedly enter wrong filenames. You should study the code below and ensure you understand each line before trying to use it – if you use it.

```
public List<BikePart> readFile(String fileName) {
    List<BikePart> retList = null;
    if (fileName == null || fileName.equals(""))
        return retList;
    File file = new File(fileName);
    try {
        retList = new ArrayList<>();
        Scanner read = new Scanner(file);
        while (read.hasNextLine()) {
            String line = read.nextLine();
            String regExp = "\\s*(\\s|,)\\s*";
            String[] pv = line.split(regExp);
            System.out.println(Arrays.toString(pv));
            BikePart bp = new BikePart(pv[0],pv[1],pv[2],pv[3],pv[4]);
            retList.add(bp);
        }
    }
    catch (FileNotFoundException e) {
        System.out.println("file not found");
        e.printStackTrace();
    }
    return retList;
}

public void writeFile(String filename, BikePart[] bpArray) {
    try{
        PrintWriter writer = new PrintWriter(fileName, "UTF-8");
        for (BikePart bp : bpArray)
            writer.println(bp); // uses BikePart toString()
        writer.close();
    } catch (IOException e) {
        System.out.println("file error!");
        e.printStackTrace();
    }
}
```

File I/O Programming

Update your program from Lab 2 so that it reads the bicycle part information from a file and writes the output to a file. Your program should terminate with informational messages.

- a message informing success

[4] The sample code may contain compile errors because I typed the code in the document.

- a message informing failure. Failure should include
    - File Not Found
    - Input file has multiple descriptions for a room

Your program should have a user interface that looks like the following.

```
Enter Input Filename: bikeParts.txt
Enter Max Cost: 25
Enter Output Filename: bikePartsLessThan25.txt
bikeparts.txt successfully processed
```

Part 3: Exceptions

You noticed file I/O used exceptions. In this part of the lab, you will learn more about exceptions, catch, and throw.

Tutorial

Today you will work through a tutorial about handling and throwing exceptions in Java. As you read, answer the questions below and write the code indicated on this sheet.

The tutorial begins here:
http://docs.oracle.com/javase/tutorial/essential/exceptions/index.html

Questions

1. What is an exception?
2. Suppose that in your code you call a method that has the potential to throw an exception. What steps must you take in your code?
3. Describe the 3 kinds of exceptions.

Write the `ListOfNumbers` class as shown in the section of the tutorial labeled "Catching and Handling Exceptions."

4. When you compile the `ListOfNumbers` class, what compilation message do you get? What does this message mean?

Keep reading to find out how to fix the syntax error.

5. What is the purpose of the try block?
6. What is the purpose of the catch block?
7. Can you include code to handle multiple exceptions associated with a single try block? Give an example.
8. What is the purpose of the finally block?
9. In the `writeList()` example in the "Putting It All Together" section, explain what happens in the `writeList()` function in each of the possible exit scenarios. At this point, "put it all together" in your implementation of the `writeList()` method, so that it compiles successfully and works as described in the tutorial.
10. When you're writing a method, how do you indicate the exceptions that may be thrown by the method?

11. What is a *stack trace*? How do you access the stack trace Information? Why is it a good idea to include information from the stacktrace when an exception is caught?
12. Complete this sentence: If a client can reasonably be expected to recover from an exception, make it a ____________________exception. If a client cannot do anything to recover from the exception, make it an ____________________ exception.
13. Name 3 advantages to using exceptions in your programs.

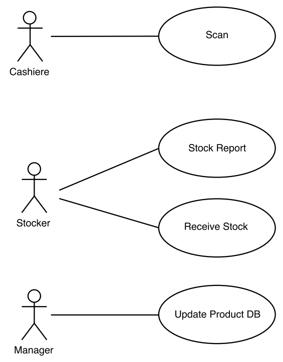
Code

Write a short main program to test the `ListOfNumbers` class.

What to Submit

- Use Case Diagram and Use Case Write Ups. These can be included in .docx or .pdf files.
- A zip file of your .java code from part 2.
- A .docx or .pdf file with the answers to questions in part 3

Use Case Diagram



Lab 4 – Classes and Class Diagrams

Tutorial

Read the OO programming tutorial.
https://docs.oracle.com/javase/tutorial/java/concepts/index.html

Questions

1. How does the tutorial define "data encapsulation"?
2. When talking about inheritance, there is a lot of terminology used to indicate the same concept. Which of the following terms mean the same thing: subclass, superclass, parent class, child class, derived class?
3. What is meant by this sentence: "Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler"?

Class Diagrams

You can use any UML drawing tool that you like; however, draw.io is the simplest to use.

- Create a class diagram for a generic `Part` class, which can be a superclass for a `BikePart` class.
- Create a class diagram for the `BikePart` class from project 1.
- Create a class diagram that shows the relationship between the `Part` class and the `BikePart` class. Include the internal elements of both classes.
- Create a class diagram for an `Inventory` class, that consists of a `BikePart` and a count of how many parts on in inventory.
- Create a class diagram for a `Warehouse` class, that consist of a collection of `Inventory`.
- Connect the class diagrams using UML connectors.

Exceptions

What to Submit

- Answers to the 3 questions and the class diagrams. Embed the diagrams in your .docx or .pdf file.

Lab 5 – Design Decisions and Tradeoffs

Work on this lab while I observe Project 1 demonstrations.

Learning Objectives

- To become familiar with alternate design decisions and the tradeoffs involved in making the decisions.
- To master terminology related to classes and object-oriented design.

Background:

- Consider the three implementations for the Day class described in Chapter 3 of the OOD&P book. These three versions represent three different design alternatives for a class whose goal is to store information about a day in time. The class aims to answer questions like "How many days exist between a specified date and the end of the year?" and "What day is 100 days in the future?"

- Feel free to consult your book and/or other CPSC 240 friends while completing the following questions. ☺

Activity

Each of the following subsections contain questions for you to answer electronically (e.g., Word, text file) or on a piece of paper. Submit answers on Canvas and bring them to class for discussion.

- Go to the book's website: http://bcs.wiley.com/he-bcs/Books?action=index&itemId=0471744875&bcsId=2561
- Click on "Browse by Resource" and then "Source Code for Programs in the Textbook" and then "Chapter 3." This will download a zip folder of all the source code for chapter 3. For this activity, you'll need the 3 folders that include the 3 versions of the Day class.

Folder day1

You can open the files in an IDE to experiment with the code.

1. What is the role of a constructor in a class? What data members are initialized by the constructor of the Day class?

2. What is the purpose of the "getter" methods of the class (getYear( ), getMonth( ), and getDate( ))?

3. Version 1 of the class includes several private methods. Will you be able to call these methods directly from your main method? (If you're not sure, try it out in an IDE and see what happens)

4. Methods that are public are said to be part of the public interface for the class. Which methods are part of Version 1 of the Day class' public interface?

5. Why would a programmer include private methods in a class?

6. In this first implementation, which methods require a large amount of effort (steps)? How would this impact the performance of a main program declaring an object of this class?

Folder day2

7. How does the implementation for this version differ from the implementation used in version1?

8. Is there any aspect of this second version of the class that is more efficient than the first? Explain.

9. Is there any aspect of this second version of the class that is less efficient than the first? Explain.

Folder day3

10. Explain one way in which version 3 is more efficient than both version 1 and 2.

11. Explain at least one reason why version 3 wouldn't always be the preferred implementation.

12. Consider the main program provided for the 3 version of the Day class. Are the main program files the same or different? Explain why.

13. When speaking of a class, what is meant by the term "public interface?"

14. When changing the implementation of a class, if you keep the public interface consistent, what advantage does this have?

What to Submit

- Answers to the questions in either .docx, .txt, or .jpeg (i.e., photo of your handwritten answers).

Lab 6 – Inheritance

Learning Objectives

- To practice object-oriented programming with inheritance.

Tutorial

Read through the inheritance tutorial from our friends at Oracle:

https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Questions

Consider the following two classes:

```
public class ClassA {
    public void methodOne(int i) {     }
    public void methodTwo(int i) {     }
    public static void methodThree(int i) {     }
    public static void methodFour(int i) {     }
}

public class ClassB extends ClassA {
    public static void methodOne(int i) {     }
    public void methodTwo(int i) {     }
    public void methodThree(int i) {     }
    public static void methodFour(int i) {     }
}
```

1. Which method overrides a method in the superclass?

2. Which method hides a method in the superclass?
3. What do the other methods do?

Code

Practice inheritance with the following classes.

- `Person` – a `Person` class contains attributes of a typical person – first name, last name, age, phone number, email address, etc.
- A `LoginAccount` class "has a" `Person`. Additionally, a `LoginAccount` class has a user name and a password.
- A `WareHouse` class has an `ArrayList<String>`, where each string represents an inventory item of bicycle part. The `WareHouse` has methods to add a string to its inventory and retrieve an `ArrayList<String>` of its inventory.
- A `SalesAssociate` class extends a `LoginAccount`. Additionally, a `SalesAssociate` class has a `String wareHouseName` and a `WareHouse`.
- A `WareHouseManager` class extends a `LoginAccount`. Additionally, a `WareHouseManager` class has a `String wareHouseName` and a `WareHouse`.

Implement a main program that creates a `SalesAssociate` and a `WareHouseManager` – both will have person attributes, user names, passwords, and a `WareHouse`. Put several `String` into the inventory of both. Create a command line user interface that prompts for a username and password, compares the values entered to `SalesAssociate` and `WareHouseManager`. If there is a match, allow a successful login after which the user can list all parts in the `WareHouse`. The following is an example run of the program.

```
Enter Username: SusanSales
Enter Password: selas
SusanSales: list
Bike part 1
Bike part 2
Bike part 3
SusanSales: logout
Enter Username:
```

Password Hashing

Java has packages that allow you to hash a password so that what is stored cannot be easily read. Use the hashing described at the following link to hash passwords in the `LoginAccount` class.

https://www.owasp.org/index.php/Hashing_Java

You will discover a method `hashPassword` that returns a hashed version of a `String` password. The signature for `hashPassword` is the following.

```
private byte[] hashPassword(final char[] password, final byte[] salt,
                            final int iterations, final int keyLength)
```

The `salt` array needs to contain random bytes. The following is an example of calling `hashPassword`.

```
Random r = new Random();
byte[] nbyte = new byte[20];
r.nextBytes(nbyte);
byte[] password;
password = hashPassword(“password”.toCharArray(), nbyte, 5, 256);
```

What to Submit

- Answers to the questions above.
- The .java files in a zip file.

Lab 7 – JavaFX

Learning Objectives

- To practice developing a JavaFX application.

Introduction

This lab will incorporate the code developed in Lab 6 into a JavaFX application. I will lead you through creating a JavaFX application.

What to Submit

- There is nothing to submit on Canvas; however, you must demonstrate to me your JavaFX application.

Lab 8 – Polymorphism and Interfaces

Learning Objectives

- To practice writing code that uses polymorphism and interfaces

Java’s Comparable and Comparator Interfaces

Read tutorials on Comparable and Comparator interfaces.

Java Comparable Interface

http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html

Java Comparator Interface

http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html

Java Comparable and Comparator Interface

https://docs.oracle.com/javase/tutorial/collections/interfaces/order.html

Part and BikePart Implementation

This section uses the `Comparable` and `Comparator` interfaces to sort `Part` and `BikePart` objects in an `ArrayList`.

Part 1: Implementing the Comparable Interface with the Part class

Start with the `Part` and `Cost` code shown below and provided on Canvas as Part.java and Cost.java. You can exclude the `Cost` class if you like.

Cost.java

```java
import java.math.BigDecimal;
import java.math.RoundingMode;
/**
 * Cost uses BigDecimal to store $ values with 2 decimal digits.
 */
public class Cost {
    private BigDecimal cost;

    public Cost(String cost) { this.cost = new BigDecimal(cost); }

    public String getCost() {
        return cost.setScale(2, RoundingMode.HALF_EVEN).toString();
    }

    public void setCost(String cost) { this.cost = new BigDecimal(cost); }

    public String toString(){
        return cost.setScale(2, RoundingMode.HALF_EVEN).toString();
    }

    public boolean equals(Cost c) {
      return cost.setScale(2, RoundingMode.HALF_EVEN).toString().equals(c.getCost());
    }
}
```

Part.java

```java
import Cost;
/**
 * Part defines a basic part which can be extended
 */
public class Part {
    protected String name;
    protected String number;
    protected Cost price;
    public Part() { }

    public Part(String name, String number, String cost) {
        this.name = name;
        this.number = number;
        this.price = new Cost(cost);
    }

    public String getName() { return name; }
    public String getNumber() { return number; }
    public String getPrice() { return price.toString(); }
    public void setPrice(String price) { this.price = new Cost(price); }
}
```

Modify the `Part` class so that it implements the `Comparable` interface. This will involve adding a method called `compareTo()` to your `Part` class and making some other changes to the class definition. Write your `compareTo()` method so that `Part` items are sorted by product name.

Now write a main program that creates an `ArrayList` of `Part`. Add about 5 `Part` object to the list so that they're randomly ordered. Print the list in its current order. Then call the Collections.sort() method to sort the `Part` objects based on name. Print the list in the new sorted order.

Part 2: Using the Comparator Interface with BikePart

Now let's work with the `Comparator` interface instead. This is the interface that doesn't require changes to the original class. Instead, you implement the `compare()` method in an external class.

Start with the `BikePart` class shown below and provided on Canvas as `BikePart.java`. Suppose that you're storing `BikePart` objects in an `ArrayList`. The `BikePart` objects need to be sorted in two ways (1) by part cost and (2) by part number.

BikePart.java

```
import Cost;
import Part;
/**
 * Part defines a bike part by extending a part
 */
public class BikePart extends Part {
    private Cost salePrice;
    private Boolean onSale;

    public BikePart(String name, String number, String listPrice,
                    String salePrice, Boolean onSale) {
        super(name, number, listPrice);
        this.salePrice = new Cost(salePrice);
        this.onSale = onSale;
    }

    public void setSale(boolean onSale) { this.onSale = onSale; }
    public boolean onSale() { return onSale; }

    public String getSalePrice() { return salePrice.toString(); }
    public void setSalePrice(String salePrice) {
        this.salePrice = new Cost(salePrice);
    }

    public void updatePriceAttributes(BikePart bp) {
        price = bp.price;
        salePrice = bp.salePrice;
        onSale = bp.onSale;
    }

    public boolean equals(BikePart bp) {
        return  salePrice.equals(bp.salePrice) &&
                onSale == bp.onSale &&
                price.equals(bp.price) &&
                name.equals(bp.name) &&
                number.equals(bp.number);
    }

    public String toString() {
        return name + ", " + number + ", " + price + ", " + salePrice +
               ", " + onSale;
    }

    public boolean getOnSale() {
        return onSale;
    }
}
```

Since we're using `Comparator`, you won't have to modify the `BikePart.java` file at all.

Make a file called `ComparatorByPartCost.java` and write the `compare()` method in that class so that `BikePart` objects are ordered from smallest to largest `Cost`. NOTE: You will have to add a `compareTo` method to `Cost`, which is used in `ComparatorByPartCost`.

Make a second file called `ComparatorByPartNumber.java` and write the `compare()` method in that class so that `BikePart` objects are ordered based on the part numbers.

Then write a main program that creates an `ArrayList` of `BikePart` objects. Add about 5 `BikePart` objects to the list in a random order. Print the list. Then call Collections.sort( ) method to sort the `BikePart` objects based on bike part cost. Print out that ordering. Then call the `Collections.sort()` method to sort the `BikePart` objects based on part number. Print out that ordering.

What to Submit

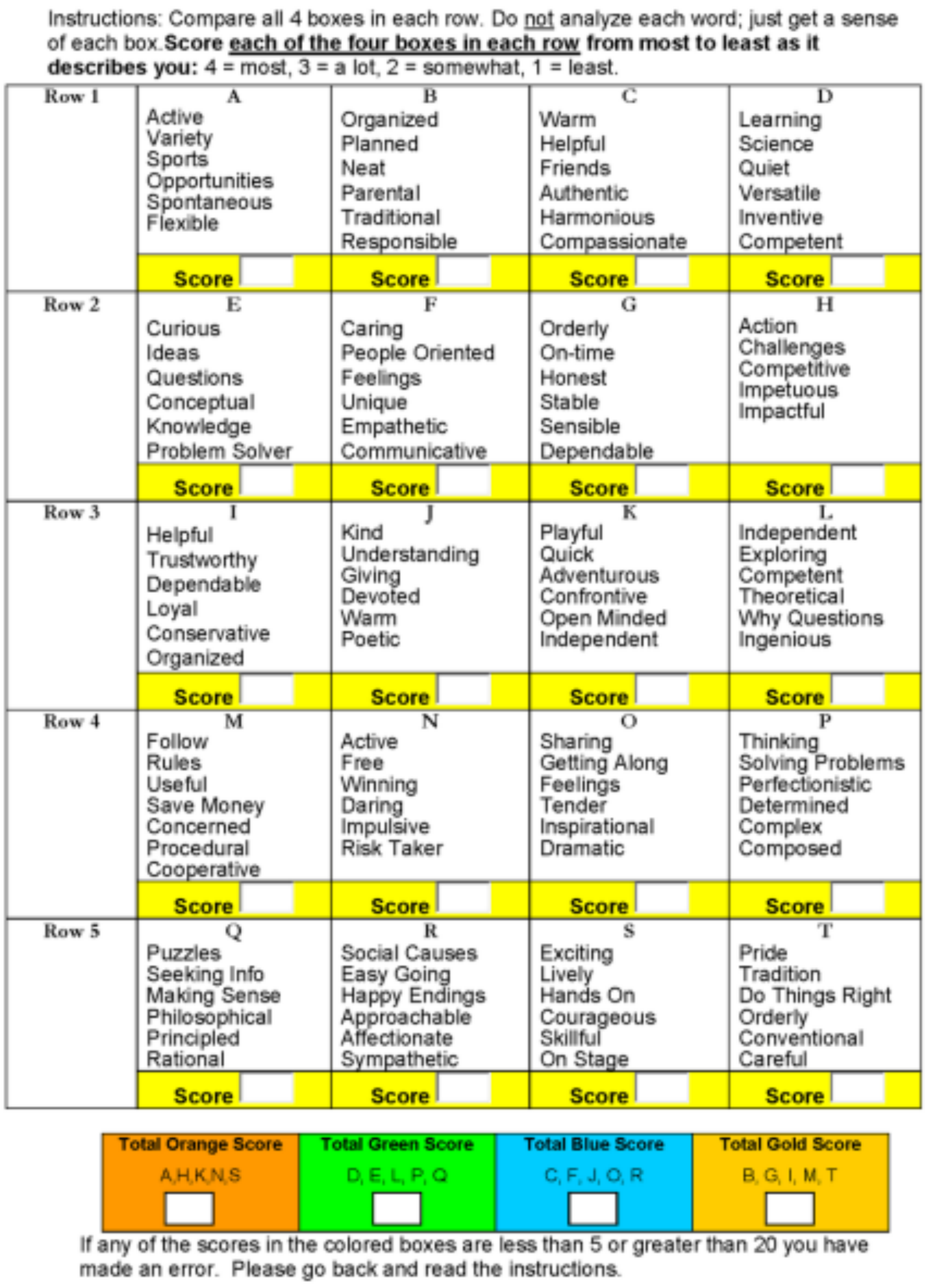
- Your .java files zipped.

Lab 9 – Group Questionnaire

Learning Objectives

- Establish your color score for constructing teams for our group project

1. Take the "True Colors" Personality Style Quiz shown below.



Instructions: Compare all 4 boxes in each row. Do not analyze each word; just get a sense of each box. **Score each of the four boxes in each row from most to least as it describes you:** 4 = most, 3 = a lot, 2 = somewhat, 1 = least.

Row 1	**A** Active Variety Sports Opportunities Spontaneous Flexible **Score**	**B** Organized Planned Neat Parental Traditional Responsible **Score**	**C** Warm Helpful Friends Authentic Harmonious Compassionate **Score**	**D** Learning Science Quiet Versatile Inventive Competent **Score**
Row 2	**E** Curious Ideas Questions Conceptual Knowledge Problem Solver **Score**	**F** Caring People Oriented Feelings Unique Empathetic Communicative **Score**	**G** Orderly On-time Honest Stable Sensible Dependable **Score**	**H** Action Challenges Competitive Impetuous Impactful **Score**
Row 3	**I** Helpful Trustworthy Dependable Loyal Conservative Organized **Score**	**J** Kind Understanding Giving Devoted Warm Poetic **Score**	**K** Playful Quick Adventurous Confrontive Open Minded Independent **Score**	**L** Independent Exploring Competent Theoretical Why Questions Ingenious **Score**
Row 4	**M** Follow Rules Useful Save Money Concerned Procedural Cooperative **Score**	**N** Active Free Winning Daring Impulsive Risk Taker **Score**	**O** Sharing Getting Along Feelings Tender Inspirational Dramatic **Score**	**P** Thinking Solving Problems Perfectionistic Determined Complex Composed **Score**
Row 5	**Q** Puzzles Seeking Info Making Sense Philosophical Principled Rational **Score**	**R** Social Causes Easy Going Happy Endings Approachable Affectionate Sympathetic **Score**	**S** Exciting Lively Hands On Courageous Skillful On Stage **Score**	**T** Pride Tradition Do Things Right Orderly Conventional Careful **Score**

Total Orange Score	Total Green Score	Total Blue Score	Total Gold Score
A,H,K,N,S	D, E, L, P, Q	C, F, J, O, R	B, G, I, M, T

If any of the scores in the colored boxes are less than 5 or greater than 20 you have made an error. Please go back and read the instructions.

Questions

1. What was your highest scoring color: Orange, Green, Blue, or Gold ?

2. If there is one person that you do not want to work with, provide the person's name. I will attempt to create groups that accommodate these requests.

3. If you listed someone in question 2, briefly explain why you do not want to work with him/her. The reason could be positive – you have worked with the person in a different course and prefer to meet new people – or negative – you do not get along.

4. If there's anything else that you think I should know when assigning groups, write it here.

What to Submit

- Answers to the questions.

Lab 10 – Unit Testing

Learning Objectives

- To practice unit testing with JUnit.
- To understand the Java Map class.

Video

Watch the video that is applicable to your IDE. If y

IntelliJ JUnit Testing

The following video shows the basics of using JUnit with IntelliJ.
https://www.youtube.com/watch?v=Bld3644bIAo

Netbeans JUnit Testing

The following video shows the basics of using JUnit with NetBeans.
https://www.youtube.com/watch?v=Q0ue-T0Z6Zs

Eclipse JUnit Testing

The following video shows the basics of using JUnit with Eclipse.
https://www.youtube.com/watch?v=I8XXfgF9GSc

Using JUnit

Select a class from your Project 1 or Project2 to create some JUnit tests. Make sure the class has several public methods that you can test. Develop a JUnit class with tests for each of the methods in your class to be tested. Think of a new method you can add to the class you are testing. Add the test case prior to creating the new method. Include comments in you JUnit class that describe your thoughts on JUnit and the idea of test driven development. For example, do you think you could develop better code following the test driven development paradigm.

Java Map

A Java `Map` is a key-value data structure. You store and retrieve values with keys. For example, I can create a Java Map that maps the name of a bike part to its corresponding `BikePart` object. When I must retrieve a `BikePart` by name, I simply use the name. I do not have to search an

`ArrayList` or array for the `BikePart` object. The following code demonstrates using a Java `Map` for storing/retrieving `BikePart` object by name.

```
private Map<String,BikePart> bpByNameMap = new HashMap<>();
BikePart bp = new BikePart("WTB_saddle","1234567890",33.00,25.58,false);
bpByNameMap.put("WTB_saddle",  bp);
bp = productMap.get("WTB_saddle");
```

Study the following an Oracle tutorial on Map to improve your understanding. You will notice this tutorial creates some maps using the Java stream API.
https://docs.oracle.com/javase/tutorial/collections/interfaces/map.html

What to Submit

- Submit the JUnit class you created.

Lab 11 – Version Control

Today's lab focuses on using GitHub, which is an online repository that you can use to share code in a group. At the end of this lab, you want to use your IDE with Git and GitHub, which will be a key component of your group project. You should already be familiar with this topic because in Writing assignment 1, you wrote a user's guide.

Learning Objectives

- To practice using Git and GitHub with Java.

Joel's Twelve Steps

Read Joel on Software "12 Steps to Better Code"
I realize this article is old, but most people find Joel's writing style amusing and his points here are still valid today.
http://www.joelonsoftware.com/articles/fog0000000043.html

Version Control

Some of you have lots of experience with version control from personal use and internship experiences. For others, it's a brand-new topic. Proceed here as suits your needs (if you're a pro, you can jump to the questions at the end. If you're new, watch one or more of the video links.)

Work through the GitHub "Hello World" Tutorial:

https://guides.github.com/activities/hello-world/

Git and GitHub References and Videos

Git and GitHub Overview
https://www.youtube.com/watch?v=xS8etehdsls

Integrating Git and GitHub into Netbeans.
https://www.youtube.com/watch?v=5yukeBNBs3w

GitHub for Beginners:
http://www.youtube.com/watch?v=0fKg7e37bQE

GitHub for Designers
If you need a little more context for the process, try this longer video:
http://www.youtube.com/watch?v=c2ygd7jOOzY

Inside GitHub
http://www.youtube.com/watch?v=YtcPn0EdUHI

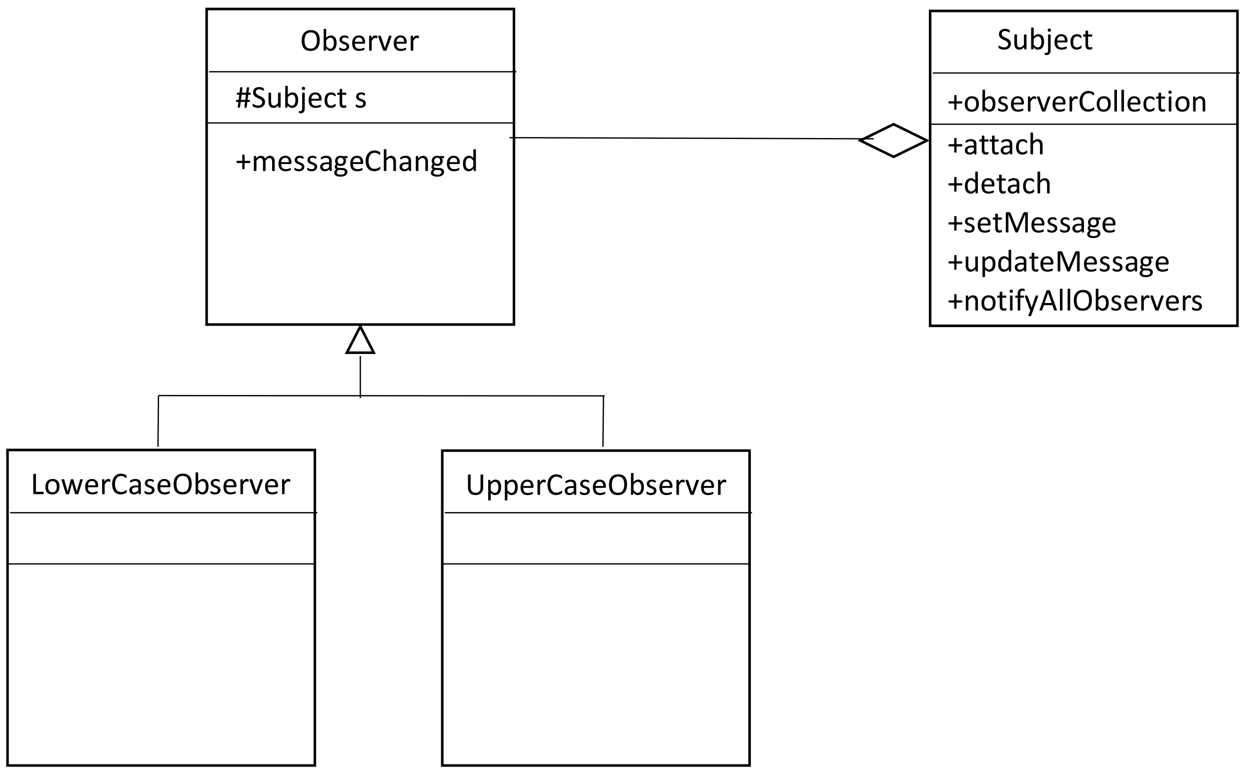
What to Submit
Do a bit of research and then turn in answers to these questions:

- What's the difference between distributed and centralized version control software?
- What features does GitHub offer that aren't found in Git?
- Have you signed up for a GitHub account?
- Which version control software do you imagine that you'll use in this course? Git is not the only option; however, it may be the easiest choice. Some additional version control software that could be analyzed are the following.
    - SVN (also known as Subversion and Mercurial).
    - Joel described CVS, which is still out there, but it's pretty "old school" and I wouldn't recommend it.
    - An option for Windows user is Team Foundation Server, which you can download for free from Imagine (the Microsoft Licensing Agreement that UMW has).

Lab 12 – Observer Pattern

Learning Objectives
- To practice Java programming with the `Observer` pattern.

Introduction
This lab creates code that implements the observer pattern. The UML for the `Observer` pattern is shown as follows. The `Subject` class has as fields (a) an observer collection, which can be a `List<Observer>` and (b) a `String message`. The `Subject` class has methods to attach and `Observer`, detach and `Observer`, set the message, update the message, and notify all observers. The `Observer` class is an abstract class that has a protected `Subject` and an abstract method `messageChanged`. The UML shows to concrete observers – `LowerCaseObserver` and `UpperCaseObserver`.



The following code implements the `Subject` class and `Observer` class from the UML, along with a `Main` that uses the `Observer` pattern.

Subject Class Code

```
import java.util.ArrayList;
import java.util.List;

public class Subject {

    private List<Observer> observers = new ArrayList<Observer>();
    private String message;

    public String getMessage() { return message; }

    public void setMessage(String message) {
        this.message = message;
        notifyAllObservers();
    }

    public void updateMessage(String message) {
        this.message += message;
        notifyAllObservers();
    }

    public void attach(Observer observer){ observers.add(observer); }

    public void notifyAllObservers(){
        for (Observer observer : observers)
            observer.messageChanged(message);
    }
}
```

Observer Class Code

```
public abstract class Observer {
    protected Subject subject;

    public abstract void messageChanged(String message);
}
```

Main Class Code

```
public class Main {
    public static void main(String[] args) {
        Subject subject = new Subject();
        Observer lc = new LowerCaseObserver(subject);
        Observer uc = new UpperCaseObserver(subject);
        subject.setMessage("HELLO hello");
    }
}
```

Code for You to Create

Implement the concrete classes – `LowerCaseObserver` and `UpperCaseObserver` – and pair them with the code provided for `Subject`, `Observer`, and `Main` such that your program successfully executes. Think of a third concrete observer and add it to your program.

What to Submit

- You .java files in a zip file.

Lab 13 – Template Method Pattern

Learning Objectives

- To improve your understanding of abstract classes and the template method pattern.

Video One and Lab Questions

Watch this video on abstract classes and answer the following questions.
https://www.youtube.com/watch?v=AU07jJc_qMQ#t=111

1. Define the term *abstract class* in your own words.

2. Can an abstract class be instantiated? For example, if we have an abstract class Animal, can we instantiate it in our main program by saying something like this?

```
        Animal fuzzyAnimal = new Animal();
```

3. Can a class be abstract even if you don't include the reserved word abstract in the class header?

4. What are some ways that abstract classes and interfaces are different?

5. What are some ways that abstract classes and interfaces are similar?

6. What is meant by the term "single inheritance"?

7. How does single inheritance fuel the need for Java to include both abstract classes and interfaces (as opposed to one or the other)?

8. Suppose that you write code (say a Cat class) that extends an abstract class (say an Animal class) and you don't override the abstract methods. What will happen when you compile your code?

9. What's the benefit to using an abstract class in a programming solution?

10. Suppose that you write code for an interface (say the interface Valuable) and include a full method implementation in the interface code. What will happen when you compile your code?

11. Suppose that you write code for an interface (say the interface Valuable) and then you implement the interface in a concrete class (say a class Gold). What happens if you don't provide the full method implementation for all of the methods specified by the interface?

Video 2 and Lab Questions

Watch this video on the Template Method Pattern and answer the following questions.
https://www.youtube.com/watch?v=aR1B8MlwbRI#t=85

PATTERN

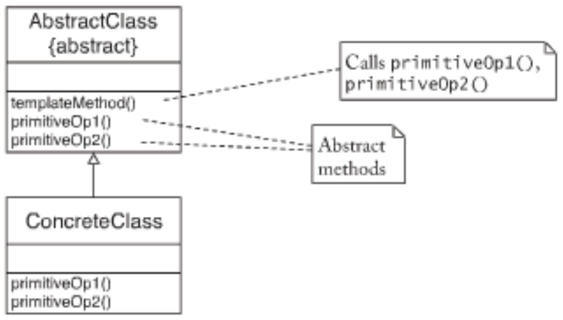
Template Method

Context

1. An algorithm is applicable for multiple types.
2. The algorithm can be broken down into *primitive operations*. The primitive operations can be different for each type.
3. The order of the *primitive operations* in the algorithm doesn't depend on the type.

Solution

1. Define an abstract superclass that has a method for the algorithm and abstract methods for the primitive operations.
2. Implement the algorithm to call the primitive operations in the appropriate order.
3. Do not define the primitive operations in the superclass or define them to have appropriate default behavior.
4. Each subclass defines the primitive operations but not the algorithm.



You should use this pattern if:

- You need to create a group of ______________________that have to execute a method that can be broken down into steps. The algorithm's steps needs to remain constant across all the data types.

- You have a set of subclass methods that are almost _______________ in implementation

- You can express the differences between the methods as another method. If you can, then move the common code into a __________________. Call the method in the __________________________for the part that varies.

1. Why is the solution where the `HoggieHut` code is implemented using the Template Method Pattern better than the first solution explored where the pattern is not used?

2. In the video, Derek shows us how to implement an `ItalianHoggie` class and a `VeggieHoggie` class. Suppose we wanted to implement a Ham & American Cheese `Hoggie` class. Write the class definition for it.

What to Submit

- Submit your answers to the questions.

Lab 14 – Iterator Pattern

Learning Objectives

- To practice the Iterator pattern and Java's ListIterator interface

Lab Work

Look at the tutorial on the Collections Interface. Pay special attention to the section on The List Interface and iterators specifically.
http://docs.oracle.com/javase/tutorial/collections/interfaces/

Explore the Java API to see the methods in the ListIterator Interface:
https://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html

Explore the How to Use Iterator tutorial:
https://www.tutorialspoint.com/java/java_using_iterator.htm

Write a program that uses the iterator with an ArrayList. Use the normal ArrayList methods as you read the data into the list. This process should look something like:

```
ArrayList myList = new ArrayList<String>();

Scanner in = new Scanner(System.in);

for (int i =0; i < 5; i++) {
   System.out.println("Enter a string");
   String data = in.next();
   myList.add(data);
}
```

Then create a list iterator object from the ListIterator class that is in the API. Use the hasNext() and next() methods to iterate through the list. As you traverse the list, retrieve and print each value from the list using the iterator.

After you've finished and tested this program, change the program so that it uses a different data structure, a LinkedList (also included in the API). Note that you should not need to change anything about your code except the line where the list itself is declared. This is the beauty of iterators. They allow you to traverse a data structure without knowing or caring about the details of the underlying list implementation.

What to Submit

Turn in your .java file.

Lab 15 – Parallelism

Learning Objectives

- Practice with Java's parallel programming techniques.

Activity 1 – Implement GreetingProducer[5]

We examine code that has two parallel threads, each printing 10 greetings. If you're having trouble understanding the ebook that goes with the parallelism topics, the OOA&D book is another helpful resource. Consider the following code for `GreetingProducer` and `ThreadTester`.

GreetingProducer Class

```
/**
   An action that repeatedly prints a greeting.
*/
public class GreetingProducer implements Runnable {
   private String greeting;
   private static final int REPETITIONS = 10;
   private static final int DELAY = 100;

   /**
      Constructs the producer object.
      @param aGreeting the greeting to display
   */
   public GreetingProducer(String aGreeting) {
      greeting = aGreeting;
   }

   public void run() {
      try {
         for (int i = 1; i <= REPETITIONS; i++) {
            System.out.println(i + ": " + greeting);
            Thread.sleep(DELAY);
         }
      }
      catch (InterruptedException exception) {
      }
   }

}
```

[5] Example and code are from the Object Oriented Analysis & Design book, Ch 9.

ThreadTester Class

```
/**
   This program runs two threads in parallel.
*/
public class ThreadTester {
   public static void main(String[] args) {
      Runnable r1 = new GreetingProducer("Hello, World!");
      Runnable r2 = new GreetingProducer("Goodbye, World!");

      Thread t1 = new Thread(r1);
      Thread t2 = new Thread(r2);

      t1.start();
      t2.start();
   }
}
```

Activity 1 Questions

1. In this program's execution, can we always guarantee that the message "1: Hello, World!" will always print before the message "2: Hello, World!"? Why or why not?

2. In this program's execution, can we always guarantee that the message "1: Hello, World!" will always print before the message "1: Goodbye, World!"? Why or why not?

Activity 2 – Implement Sum Values Algorithm

Implement the algorithm to sum the values in an array of threads that we've been working on in class. The following subsections contain the pseudocode and classes.

Pseudocode

```
int sum(int[] arr) {
  results = new int[4];
  len = arr.length;
  FORALL(i=0; i < 4; ++i) { // parallel iterations
     results[i] = sumRange(arr,(i*len)/4,((i+1)*len)/4);
  }
  return results[0] + results[1] + results[2] + results[3];
}
int sumRange(int[] arr, int lo, int hi) {
   result = 0;
   for(j=lo; j < hi; ++j)
      result += arr[j];
   return result;
}
```

SumThread Class

```
class SumThread extends java.lang.Thread {
    int low; int hi; int[] arr;
    int ans = 0;
    SumThread(int [] a, int l, int h) {
         low = l; hi = h; arr=a;
    }

    public void run() {
        for (int i = low; i < hi; i++) {
            ans += arr[i];
        }
    }
}
```

Main Class

```
class Main {
   public static void main (String args[]) throws InterruptedException{
       int array[] = new int[10];
       for (int i = 1; i < array.length; i ++) {
           array[i] = i;
       }
       int result = sum(array);
           System.out.println("RESULT = " + result);
    }
    public static int sum(int[] arr) throws InterruptedException {
        int len = arr.length;
        int ans = 0;
        SumThread[] ts = new SumThread[4];
        for (int i = 0; i < 4; i++) {
            ts[i] = new SumThread(arr, i*len/4, (i+1) * len/4);
                        ts[i].start();
        }
        for (int i = 0; i < 4; i++) {
            ts[i].join();
            ans += ts[i].ans;
        }
        return ans;
    }
}
```

Activity 2 Questions

3. What changes are made in the code above compared to the code from our class notes?

What to Submit

- Answers to the questions

Projects Overview

Projects are an important component of CPSC 240. Projects contain more classes and code than labs. Projects allow us to practice our object-oriented programming skills on real-world-like problems.

CPSC 240 has three projects that are connected, which means that code you create in Project 1 can be used in Project 2 and code you create in Project 2 can be used in Project 3.

CPSC 240 projects develop code for a bicycle parts distributorship.

Project 1 uses text input and output from the terminal. Projects 2 and 3 have a JavaFX GUI.

Programming Projects

- 1BicyclePartsDistributorship.docx
- 2BicyclePartDistributorship.docx
- 3BicyclePartDistributorship.docx

Project 1 – Bicycle Parts Distributorship

This project is the first of three connected projects. The first two are individual projects. The third is a group project. You should design your code such that you can continue adding to it.

NOTE: The Bicycle Parts Distributorship project specifications use the term database as a general description of the underlying data that must be persistent across executions of your program. To satisfy the persistent requirement, your database will have to be a file (or files) that your program reads/writes. You do not have to create an SQL or NOSQL database.

Introduction

You are a programmer working for Bicycle Parts Distributorship. BPD has a warehouse full of bicycle parts. Each bicycle part has the following attributes.

- Part name - example names are spoke, saddle, brake pads, and tires
- Part number - a part number is a 10-digit number
- List price - this is what the part costs normally
- Sale price - this is what the part costs when on sale
- On sale - When this is false, the list price applies, otherwise the sale price applies

In addition to the bicycle parts attributes, the warehouse has an attribute indicating the number of parts in its inventory.

- Quantity - this is the number of parts that Bicycle Parts Distributorship has in its warehouse.

You shall write a program that maintains an inventory of bicycle parts and allows sales associates to sell bicycle parts.

Warehouse Inventory Update

When inventory is delivered to the warehouse, it is accompanied by a text file describing parts in the delivery. The database of parts in the warehouse is created and updated by reading the inventory delivery file. The file consists of a sequence of comma-separated lines. Each line describes the attributes of a specific auto part. The attributes on each line of the file match those provided in the Introduction section in the following order.

```
partName,partNumber,listPrice,salePrice,onSale,quantity
```

The quantity indicates the number of parts in the delivery. The following is an example inventory update file.

```
WTB_saddle,1234567890,33.00,25.58,false,25
26inTube,1234567891,7.00,5.58,true,25
seatPost,1234567892,17.00,15.21,true,25
carbonHandleBars42cm,1234567893,47.00,5.58,true,25
700-23SwhalbeTire,1234567894,50.00,40.50,true,10
700-25SwhalbeTire,1234567895,50.00,40.50,true,10
10spRearDerailuer,1234567896,60.00,50.50,true,10
10spFrontDerailuer,1234567897,60.00,50.50,true,10
```

```
11spRearDerailuer,1234567898,60.00,50.50,true,10
11spFrontDerailuer,1234567899,60.00,50.50,true,10
mensBibsMedium,1234567900,110.00,99.00,false,4
```

Program Functional Requirements

Your program shall accomplish the following.

1. Your program shall have a warehouse database in the file warehouseDB.txt. The first time your program runs the warehouseDB.txt file shall be empty or nonexistent.
2. Your program shall read the file warehouseDB.txt upon starting. This shall establish the initial inventory in your internal data structures.
3. Your program shall write the internal data structures to warehouseDB.txt upon exiting. The inventory in your internal data structures shall be saved such that upon running your program again, it starts with the same data as it ended with.
4. Your program shall read inventory delivery files (e.g., inventory.txt) as input by the user. Reading inventory delivery files updates the data in your internal data structures as follows.
    a) If the bicycle part is not a member of your internal data structure, the bicycle part is added.
    b) If the bicycle part is is in your internal data structure, the list price is updated, the sales price is updated, the on sale indicator is updated, and the quantity is added to the current quantity on hand.
5. Your program shall show bicycle part information by name.
6. Your program shall sell a bicycle part by part number. When a bicycle part is sold, you shall do the following.
    a) Display the part name, the part cost, which will be either the list price of sale price.
    b) Display the time the part was sold.
    c) Decrement the quantity.
7. Your program shall allow a new part to be entered interactively.
8. Your program shall display all parts in alphabetical order.
9. Your program shall display all parts in part number order.

User Interface

This section describes a command line user interface and connects the user interface to the program requirements.

```
Please select your option from the following menu:
Read: Read an inventory delivery file
Enter: Enter a part
Sell: Sell a part
Display: Display a part
SortName: Sort parts by part name
SortNumber: Sort parts by part number
Quit:
Enter your choice:
```

Each choice requires you to prompt for additional information, gather the user's information, and then perform the action.

- `Read`: - prompt for a filename of the inventory delivery file. Read the file and update your internal data structures as described in the program requirements. If the file is not present, post an error message.
- `Enter`: - prompt for a part name, part number, list price, sale price, on sale, and quantity. After gathering the user's input, add the new bicycle part to your internal data structure
- `Sell`: - prompt for a part number. After gathering the part number,
  - Display the part name, the part cost, which will be either the list price of sale price.
  - Display whether the part is on sale.
  - Display the time the part was sold.
  - Decrement the quantity of the part in your internal data structures.
  - If the part number is not in your data structure, display an error message.
- `Display`: - prompt for a part name. After gathering the part name,
  - Display the product name, the product cost, which will be either the list price of sale price.
  - Display the time the product was sold.
  - Decrement the quantity.
  - If the part name is not in your data structure, display an error message.
- `SortName`: - display a list of parts sorted by part name.
- `SortNumber`: - display a list of parts sorted by part number.
- `Quit`: - quit your program and write the internal data structure to the file inventory.txt so that the next time your program runs, it resumes with the inventory as saved from the previous run.

What to Submit

Project Plan

Identify the classes that you will include in your project. For each class, provide the class name, a description of each of its data members, and a description of each of its methods. Turn this information in to Canvas.

Final Project

- Demonstrate your project running on your computer in class on demonstration day.
- Turn in your files electronically to Canvas. Include all .java files for your project. Place them in a .zip file prior to uploading them.
- To Canvas, turn in a one-page explanation of how you tested your project to ensure that it always provides the expected output. If it doesn't always work correctly, note the cases where it does not provide the expected behavior.

Criteria for Program Evaluation

- Correctness of answers and form of output
- Documentation (including class headers, method headers, and in-line commentary)
- Quality of object-oriented design

- Program structure and design including: program organization, maintainability, portability, reliability, readability, robustness, efficiency, use of language, and satisfaction of all requirements as stipulated in the assignment.

Rubric

Criteria	Points	Score
Project Compiles - Having a compiling project doesn't earn you any points, but if your project doesn't compile, you earn a zero on the project.	0	
An individual bicycle part can be added to a warehouse inventory.	10	
The main warehouse can be updated with an inventory delivery file.	10	
List of bicycle parts can be printed in part name order; list is formatted neatly	10	
List of bicycle parts can be printed in part number order; list is formatted neatly	10	
A bicycle part can be sold	10	
User can quite the program, and when user restarts the program, the warehouse inventory is that which existed when the user quit	10	
All required information is stored for each part	10	
Code includes JavaDoc in every file that includes the author's name & a brief description of the class	5	
Code includes good Javadoc method header comments associated with every method in every class	5	
Code demonstrates good design principles as discussed in class	5	
Code is turned in as a zip file	5	
Project plan was submitted on time and correct	10	

File Input and Output[6]

There are two types of files in Java - text files and binary files. Files provide both sequential and random access. A text file is processed as a sequence of characters. A binary file is processed as a sequence of bytes. In a text file you have the illusion that the file is divided into lines. There is a special **end-of-line** symbol that creates this illusion. In addition you can think that there is a special **end-of-file** symbol that follows the last component in a file. A big advantage of text files is their portability. In binary files, the representation used varies from computer to computer. Java binary files are platform independent. They can be interpreted by any computer that supports Java.

[6] http://www.cs.utexas.edu/~mitra/csSummer2009/cs303/lectures/fileIO.html

A stream is a device for transmitting or retrieving 8-bit or byte values. A file is a collection of items stored on an external device. The Java object FileStream provides the means to access the data values but does not actually hold the file contents.
There are two independent and largely parallel systems involved in I/O. *InputStream* and *OutputStream* are used to read and write 8-bit quantities and process binary files. The alternative hierarchy has two different classes Reader and Writer that are used to read and write 16-bit Unicode character values and process text files.

The *File* class provides methods for dealing with files or directories. File systems are organized into a hierarchy. A path is a description of a file's location in the hierarchy. When a program is running, the program' directory is considered the current directory. Any files located in the current directory can be referred to by name alone. The relative path is the location of a file with respect to the current directory. The absolute path starts at the root directory.

```
public class File extends Object implements Serializable {
// Constructors
public File ( String path );
public File ( String path, String name );
// Public methods
public boolean canRead();     // is the file readable?
public boolean canWrite();    // is the file writeable?
public boolean delete();      // delete the file
public boolean exists();      // does the file exist
public long lastModified();   // when was the file last modified
public long length();         // How many bytes does it contain   public
boolean renameTo(File f);   // rename this file to f's name
}
```

Creating a Handle to a File

A handle to a file is created by passing the name of the file to the constructor for the File object:

File inFile = new File ("FileIO.txt");

Reading from a Text File

A text file can be read using a Scanner object. Using the Scanner offers the advantage of using the methods that come with the Scanner class.

```
import java.util.Scanner; import java.io.*;
public class ReadTextFile {
   public static void main (String [] args) throws IOException    {
      File inFile = new File ("input.txt");
      Scanner sc = new Scanner (inFile);
      while (sc.hasNextLine())     {
         String line = sc.nextLine();
         System.out.println (line);
      }
      sc.close();
   }
}
```

Writing to a Text File

To write text to a file you open an output stream by using the class *FileWriter*. If the file does not exist a new empty file with this name is created. If the file already exists opening it erases the data in the file. If you want to append to the file use the following option when creating the FileWriter object:

FileWriter fWriter = new FileWriter (outFile, true);

The class *PrintWriter* has methods print(), printf() and println() that will allow us to write to a file.

```
import java.io.*;
public class WriteTextFile {
   public static void main (String [] args) throws IOException   {
     File outFile = new File ("output.txt");
    FileWriter fWriter = new FileWriter (outFile);
     PrintWriter pWriter = new PrintWriter (fWriter);
     pWriter.println ("This is a line.");
     pWriter.println ("This is another line.");
     pWriter.close();
   }
 }
```

Project 2 – Bicycle Parts Distributorship

This project is the second of three connected projects. The first two are individual projects. The third is a group project. You should augment your design and code from Project 1 to include the new capabilities described in this project description.

NOTE: The Bicycle Parts Distributorship project specifications use the term database as a general description of the underlying data that must be persistent across executions of your program. To satisfy the persistent requirement, your database will have to be a file (or files) that your program reads/writes. You do not have to create an SQL or NOSQL database.

Introduction

You are a programmer working for Bicycle Parts Distributorship. BPD has a warehouse full of bicycle parts. The warehouse parts and database are as described in Project 1. In this increment of the project BPD has purchased vans and hired sales personnel who drive the vans. The sales personnel load their vans with bicycle parts, which they sell to bicycle shops located throughout the sales district. Each sales van is its own warehouse that has inventory. When a sales van is loaded with parts, the inventory in the main warehouse is decremented and the inventory in the sales van is incremented. Sometimes, a sales associate does not want to drive to the main warehouse to get a few parts. In these cases, sales associates agree to meet and move parts from one sales van warehouse to another.

Warehouse Parts Inventory Update

The warehouse parts inventory update shall be done by processing an inventory delivery file just like Project 1.

Sales Van Parts Inventory Update from Main Warehouse

When a sales associate visits the warehouse to load their van with parts, the sales associate arrives with a sales van delivery file that describes the bicycle parts that are to be moved from the main warehouse onto their van. The file consists of a sequence of comma-separated lines. The first line contains the name of the main warehouse followed by the name of a destination van, where parts are moved from the main warehouse to the destination. Each of the remaining lines contains the name of the bicycle part and the quantity that is to be moved from the main warehouse into the sales van warehouse. The following is an example.

```
mainWarehouse,salesVanC
WTB_saddle,4
26inTube,10
seatPost,2
carbonHandleBars42cm,25
```

Sales Van Parts Inventory Update from a Sales Van

When sales associate A meets with a sales associate B to get parts, sales associate A arrives with a sales van delivery file that describes the bicycle parts that are to be moved from sales associate B's van into sales associate A's van. The file consists of a sequence of comma-separated lines. The first line contains the name of the source van followed by the name of a destination van, where parts are moved from the source to the destination. Each of the remaining lines contains the name of the bicycle part and the quantity that is to be moved from the source van to the destination van. The following is an example.

```
salesVanA,salesVanB
WTB_saddle,4
26inTube,10
seatPost,2
carbonHandleBars42cm,25
```

When a sales van delivery file is processed in this case, the inventory in both the salesperson A's delivery van warehouse and salesperson B's delivery van warehouse are updated.

Program Functional Requirements

Your program shall accomplish the following. NOTE: Requirements 1 through 9 are identical from Project 1; however, some implementation details such as filenames have been removed.

1. Your program shall have a warehouse database. The first time your program runs the warehouse database shall be empty or nonexistent.
2. Your program shall read the warehouse database upon starting. This shall establish the initial inventory in your internal data structures.
3. Your program shall write the internal data structures to the warehouse database upon exiting. The inventory in your internal data structures shall be saved such that upon running your program again, it starts with the same data as it ended with.
4. Your program shall read inventory delivery files (e.g., inventory.txt) as input by the user. Reading inventory delivery files updates the data in your internal data structures as follows.
    a) If the bicycle part is not a member of your internal data structure, the bicycle part is added.
    b) If the bicycle part is is in your internal data structure, the list price is updated, the sales price is updated, the on sale indicator is updated, and the quantity is added to the current quantity on hand.
5. Your program shall show the bicycle part information by name.
6. Your program shall sell a bicycle part by part number from a warehouse. When a bicycle part is sold, you shall do the following.
    a) Display the part name, the part cost, which will be either the list price of sale price.
    b) Display the time the part was sold.
    c) Decrement the quantity.
7. Your program shall allow a new part to be entered interactively.
8. Your program shall display all parts in alphabetical order for
    a) The total collection of warehouses
    b) The main warehouse
    c) Each individual sales van warehouse
9. Your program shall display all parts in part number order for
    a) The total collection of warehouses
    b) The main warehouse
    c) Each individual sales van warehouse

10. Your program shall allow the user to add a sales van to the fleet.
11. Your program shall allow a sale van to move inventory from the main warehouse into the sales van warehouse.
12. Your program shall allow a sales van to move inventory from another sales van warehouse into its sales van warehouse.

Object-oriented Requirements

1. Your program shall define and use a Java Interface as part of its solution.
2. Your program shall define and use inheritance as part of its solution.
3. You shall create the following UML constructs.
    a) Use case diagrams.
    b) Class diagrams

User Interface Requirements

1. Your program shall have a JavaFX Graphical User Interface. The layout of the GUI panels is your design, but I recommend you create a simple interface.

What to Submit

Project Plan

Identify the classes that you will include in your project. For each class, provide the class name, a description of each of its data members, and a description of each of its methods. Turn this information in to Canvas.

Final Project

- Demonstrate your project running on your computer in class on demonstration day.
- Turn in your files electronically to Canvas. Include all .java files for your project. Place them in a .zip file prior to uploading them.
- To Canvas, turn in a one-page explanation of how you tested your project to ensure that it always provides the expected output. If it doesn't always work correctly, note the cases where it does not provide the expected behavior.

Criteria for Program Evaluation

- Correctness of answers and form of output
- Documentation (including class headers, method headers, and in-line commentary)
- Quality of object-oriented design
- Program structure and design including: program organization, maintainability, portability, reliability, readability, robustness, efficiency, use of language, and satisfaction of all requirements as stipulated in the assignment.

Rubric

Criteria	Points	Score
Project Compiles - Having a compiling project does not earn points, but if your project does not compile, you earn a zero on the project.	0	

Code is submitted as a zip file – Not submitting a zip file results in negative points.	-5	
An individual bicycle part can be added to a warehouse inventory.	10	
The main warehouse can be updated with an inventory delivery file.	10	
List of bicycle parts can be printed in part name order; list is formatted neatly	10	
List of bicycle parts can be printed in part number order; list is formatted neatly	10	
A bicycle part can be sold	10	
User can quit the program, and when user restarts the program, the warehouse inventory is that which existed when the user quit	10	
All required information is stored for each part	10	
Code includes JavaDoc in every file that includes the author's name & a brief description of the class	5	
Code includes good Javadoc method header comments associated with every method in every class	5	
Code demonstrates good design principles as discussed in class	10	
Project plan was submitted on time and correct	10	

Project 3 – Group Project – Bicycle Parts Distributorship

This project is the third of three connected projects. The first two are individual projects. The third is a group project. Your group has several implementations of Project 2. Your group should augment the design and code from one (or more) of the Project 2 implementations to include the new capabilities described in this project description.

NOTE: The Bicycle Parts Distributorship project specifications use the term database as a general description of the underlying data that must be persistent across executions of your program. To satisfy the persistent requirement, your database will have to be a file (or files) that your program reads/writes. You do not have to create an SQL or NOSQL database.

Project Tips from Previous Students

- You should emphasize the importance of good time management. **Start early!** When the group project is assigned, you may think there is plenty of time. Do not delay, thinking you will catchup in a couple of weeks.
- I recommend that teams perform an agile software development process, which is a sequence of sprints where each sprint incrementally adds code to your project. Divide the software capabilities into small related collections, which can be incrementally developed. You begin by developing the first collection of capabilities, where development includes testing. Then you develop each subsequent collection of capabilities. At a minimum, you should push your code to GitHub after each sprint.
- Several of our classes have a portion of time dedicated to project group work; however, you should plan to meet outside of class. These meetings can be in-person or via Skype/Facetime. The most successful teams are those that communicate a lot, and hold each other accountable.
- Usually, the most successful teams meet in-person and perform pair/partner programming with ongoing discussions. This style of development results in less hassle trying to integrate your code in the end and if you get stuck you have group members right there to work through it.
- If you divvy your programming into three chunks with minimal group meetings, integrating the separate chunks into a working program will be challenging. You should plan for at least three days to all the code together at the end.
- Have multiple people contribute to GUI development. Do not assign it to a one person.
- Using Serializable object I/O is a convenient way to read/write your integral data structures to a "database" file.

Tips for Successfully Working in a Group

The following tips for successfully working are an edited version of those found in http://www.cs.cmu.edu/~pausch/Randy/tipoForGroups.html.

- **Meet people properly**. Introduce each other. Make sure you can pronounce each other's names. Exchange contact information (email, phone numbers) and establish acceptable times for contact each other.
- **Find things in common**. Discover something in common with each. Having common interests will make it easier to address issues where you have differences. At a minimum,

you like computer science and experience the same weather. Maybe you like the same sports or music or movies.

- **Establishing Meetings** – Ensure everyone knows meeting dates and are accountable for being prepared. This can be done by email. Do not assume roommates deliver phone messages.
- **Electronic Communication** – Include all team members on electronic communication. This ensures everyone is kept up-to-date.
- **Written Records.** Keep a record of tasks and who is responsible for what and by when.
- **Hold Meetings in Nice Places**. Meet in a comfortable area (e.g., HCC or Trinkle lab) with appropriate internet connection with surfaces to accommodate laptops and/or notebooks. Long meetings may need snacks or coffee.
- **Let Everyone Talk**. Listen when someone is talking, even if you do not like what they are saying. Cutting someone off is rude, and not worth whatever small time gain you might make. Do not finish someone's sentences. And remember: talking louder or faster does not make your idea any better.
- **Check Egos at Door**. Record ideas with a descriptive label and not the originator, e.g., "the troll bridge story," not "Jane's story."
- **Complement Each Other**. Focus on the good, complement it, and then raise any objections or concerns you have about the rest of it.
- **Be Open and Honest**. Discuss problems with group members, and ask me if you need help. Be forgiving of mistakes, but address issues when they come arise.
- **Avoid Conflict**. When stress occurs and tempers flare, take a short break, clear your heads, apologize, and start again. Apologize for upsetting your teammates, even if you think someone else was at fault.
- **Phrase alternatives as questions**. When your group is considering option A, it is better for you to say, "*What if we tried B?*" than to say "*I think we should do B.*"

Effective Meetings

Your group will meet to analyze the project, create the specification for Writing 3, create a design, implement code, test the system, and develop your presentation. Meetings are most effective when the adhere to the following simple rules.

- Establish an agenda for the meeting – the agenda lists the objectives of the meeting and keeps the meeting on focus. Any actions from previous meetings should be on the agenda.
- Start on time and end on time.
- End the meeting with actions – An action should have the following attributes.
    - A description of the action
    - The person(s) to whom the action is assigned.
    - The date the action is to be completed.

Tools for Collaboration

- Trello.com – allows you to define tasks, assign to individuals, and track using features such as backlog, in-progress, and completed.
- Slack.com – allows groups to send messages to all in the group.
- Google Docs – allows groups to share documents.

- Phone Calls or Text Messages

Introduction

You are a programmer working for Bicycle Parts Distributorship. BPD has a warehouse full of bicycle parts. The code and underlying data structures from Projects 1 and 2 apply to Project 3, but you will have to modify them. In this increment of the project BPD has identified several users of the software. There is an office manager who keeps track of many business aspects, there is a warehouse manager, and there is one sales associate in each sales van. In your program, these employees (office manager, warehouse manager, and sales associates) have accounts that allow them to perform actions that are specific to their duties. The duties that each type of employee performs are described in various use case scenarios. Additionally, there is a system administrator who manages accounts.

System Administrator Use Case Scenarios

A system administrator must be logged in to perform use case scenarios described in the following subsections.

The system administrator is responsible for creating accounts, deleting accounts, and resetting passwords. An account shall include such things as user, phone number, email, username, and password. Your program shall have default system administrator account that has the following.

- Username: admin
- Password: minda

When a system administrator creates an account, the initial login credentials shall be as specified by the GUI panel, which will allow various information such as the following.

- First Name
- Last Name
- email
- Username
- Password

Optional: Users shall have the ability to change their password and user information such as phone number and email; however, users cannot change their usernames.

Add Office Manager Use Case Scenario

The system administrator shall add an office manager account to the system.

Add Main Warehouse Manager Use Case Scenario

The system administrator shall add a warehouse manager account to the system.

Add Sales Associate and Sales Van Use Case Scenario

Project 2 allowed you to add a sales van. In Project 3, the system administrator shall be responsible for adding a sales associate and a sales van. This connects a sales associate to a sales van. For example, Betsy Soho has SalesVanA. This use case scenario establishes a sales associate account for each sales associate along with the sales van for the sales associate.

Office Manager Use Case Scenarios

The office manager has several use case scenarios described in the following subsections. An office manager must be logged in to perform these scenarios.

Examine Part Use Case Scenario

The office manager shall have the ability to examine the information for a part by the following.

- Enter a part name.
    - **Optional**: Include a wild card as part of the match. The wild card allows matching parts that are similar. For example, you may choose to allow sa* to match saddle and saw.
- Enter a part number.
- **Optional**: Enter a quantity. This examine request allows the office manager to include less than, equal to, or greater than, where the display shows all parts that satisfy the relational operator applied to the quantity. For example, the office manager may use this option to display all parts that have more than 10 in all warehouses.

Order Parts Use Case Scenario

Each bicycle part shall have a minimum amount that BPD warehouse wants to always have in stock. When the total stock falls below the minimum amount, the office manager is notified to generate an order to refill the part. Since it is inefficient to order one part, the office manager shall have the ability query the warehouse inventory to determine all parts that are either below their minimum or close to the minimum. The office manager shall use below minimum notifications and the inventory order query to determine how many new parts to order.

Sales Commission Use Case Scenario

Sales associates are paid on commission - they receive a salary that is 15% of their total sales. The office manager shall have the ability to generate a pay check for each sales associate. The office manager shall input a sales associate, a start date, and an end date. The output shall be the amount to be paid to the sales associate for the period selected.

Warehouse Manager Use Case Scenario

The warehouse manager has several use case scenarios described in the following subsections. An office manager must be logged in to perform these scenarios.

Warehouse Inventory Update Use Case Scenario

The warehouse manager shall be responsible for performing the Warehouse Parts Inventory Update as described in Projects 1 and 2.

Examine Part Use Case Scenario

The office manager shall have the ability to examine the information for a part by the following.

- Enter a part name.

    - **Optional**: Include a wild card as part of the match. The wild card allows matching parts that are similar. For example, you may choose to allow sa* to match saddle and saw.
- Enter a part number.
- **Optional**: Enter a quantity. This examine request allows the office manager to include less than, equal to, or greater than, where the display shows all parts that satisfy the relational operator applied to the quantity. For example, the office manager may use this option to display all parts that have more than 10 in all warehouses.

Sales Associate Use Case Scenario

Sales associates have several use case scenarios described in the following subsections. A sales associate must be logged in to perform these use case scenarios.

Sales Van Loading Use Case Scenario

A sales associate must keep appropriate parts on their sales van to keep their bike shop customers adequately stocked. A sales associate generates a sales van delivery file that is used when loading parts onto their sales van. The requirements for this use case scenario are as described in Project 2, Section Sales Van Parts Inventory Update from Main Warehouse and Section Sales Van Parts Inventory Update from a Sales Van; however, for this increment each sales associate can only load their own sales van.

Sales Invoice Use Case Scenario

When a sales associate distributes parts to a bike shop, the sales associate selects that parts to be sold. The sales associate shall select a part name (or number) and the quantity be sold. The sales associate shall select as many parts as the bike shop wants to purchase. The sales associate shall generate a sales invoice that is a list of parts sold that includes the name of the bike shop employee who received the parts. The invoice shall show the part name, part number, the number of parts sold, the list price, the sales price, and the total cost. The invoices shall be stored in the BPD database. The office manager shall use the invoices to compute the commission of each sales associate. The following is an example invoice.

```
Sales Invoice for Susan's Bicycle Shop, September 21, 2017 at 10:15AM
Part Name           Part Number  Price  Sales Price Qnty    Total Cost
WTB_saddle          1234567890   33.00        33.00   3        99.00
26inTube            1234567891    7.00         5.58   1         5.58
seatPost            1234567891   17.00        15.21   2        30.42
carbonHandleBars42cm 1234567891  47.00        47.00   1        47.00
Total                                                         182.00

Received By Signature: Susan the Owner
```

Program Functional Requirements

Your program shall accomplish the following.

13. Your program shall have a warehouse database. The first time your program runs the warehouse database shall be empty or nonexistent.

14. Your program shall read the warehouse database upon starting. This shall establish the initial inventory in your internal data structures.
15. Your program shall write the internal data structures to the warehouse database upon exiting. The inventory in your internal data structures shall be saved such that upon running your program again, it starts with the same data as it ended with.
16. Your program shall have accounts for several types of users. Each user shall be logged into the system to perform their actions.
    1. An administrative account, which shall perform the system administrator use case scenarios.
    2. An office manager account, which shall perform the office manager use case scenarios.
    3. A warehouse manager account, which shall perform the warehouse manager use case scenarios.
    4. Several sales associate accounts, which shall perform the sales associate use case scenarios.

Object Oriented Requirements

The last section of this document provides a class diagram of Gusty's solution. You are welcome to use any (or all) of the design.

1. Your program shall at least two Java packages
2. Your program shall define and use a Java Interface as part of its solution.
3. Your program shall define and use inheritance as part of its solution.
4. Your program shall use at least two of the patterns we studied.
5. Your program shall include JUnit testing for at least two of its classes.
6. You shall create the following UML constructs.
    c) Use case diagram that shows all use cases. You can reuse work from Lab 3
    d) Class diagrams – The class diagrams do not have to include fields and methods. I want a diagram that shows the relationships between your classes.

User Interface Requirements

1. Your program shall have a JavaFX Graphical User Interface. You may create the layout of the GUI panels.
2. When a user is logged in, the user only has access to features allowed for their role as described in the use case scenarios above. For example, a sales associate cannot generate their sales commission because that is a feature of an office manager account.

Group Project Progress Check

To earn the points for this check, you need to make a substantial contribution to the code on your group's GitHub repository before this date.

What to Submit for Group Progress Check

Submit a paragraph describing your code contribution and a link to the GitHub repository for the group.

Individual Reflection and Peer Review

For the reflection, write a paper reflecting on your experience with the group project. The paper shall include the following.

- A paragraph (or two) describing your contributions to the project. Highlight anything significant that you did to help with the project (technical or non-technical). Be sure to provide implementation details about the most significant contribution that you made to the code. If it was a full class, provide an overview. If it was a method, provide details about where the method is found and how it works with the rest of the group's code. Let me know how these features correspond to your GitHub activity.
- A paragraph (or two) on each team member that describes the positive contributions your team member made to the project.
- A paragraph (or two) on each team member that describes what you wish your team member had done differently. You may think a team member did everything that should have been done. If so, state this.
- A paragraph (or two) describing what you would do differently if you redid the project.

Also include a response to this question:

Your boss is so impressed with your project, that she's awarding your team $3000 in bonuses. Decide how to divide them up

You:	Amount: ______________________________
Name 1: __________________________	Amount ______________________________
Name 2: __________________________	Amount ______________________________

Peer Review

You will receive a Peer Review grade in Canvas. Your peer review grade will be based on the peer reviews of you by your groups members' individual reflections.

Project Demonstration

Your group's presentation and demonstration should be approximately 15-20 minutes long. The main point of the presentation is to demonstrate the features that you have working in the final implementation. The projects will be graded with the rubric found in this assignment,

Topics that you should cover in the presentation are:

1. Who are you building the software for and why does your version of Twitter meet their needs
2. What features did you include? Show use case diagram/s as needed.
3. How did you approach the design?
   a. Explain the big picture classes included in the final code. Show class diagram/s as needed.
   b. Explain how you satisfied the object-oriented requirements.
4. Demo the software to show off all the functionality

Each group member shall participate in the presentation and demonstration.

What to Submit

- Presentation and demonstration in class during the last week of classes. Each group member shall participate in the presentation and demonstration.
- Link to your GitHub repository.
- Zipped folder of your code. Only one team member must submit the zip file.
- Individual reflection and peer review described previously.

Grading

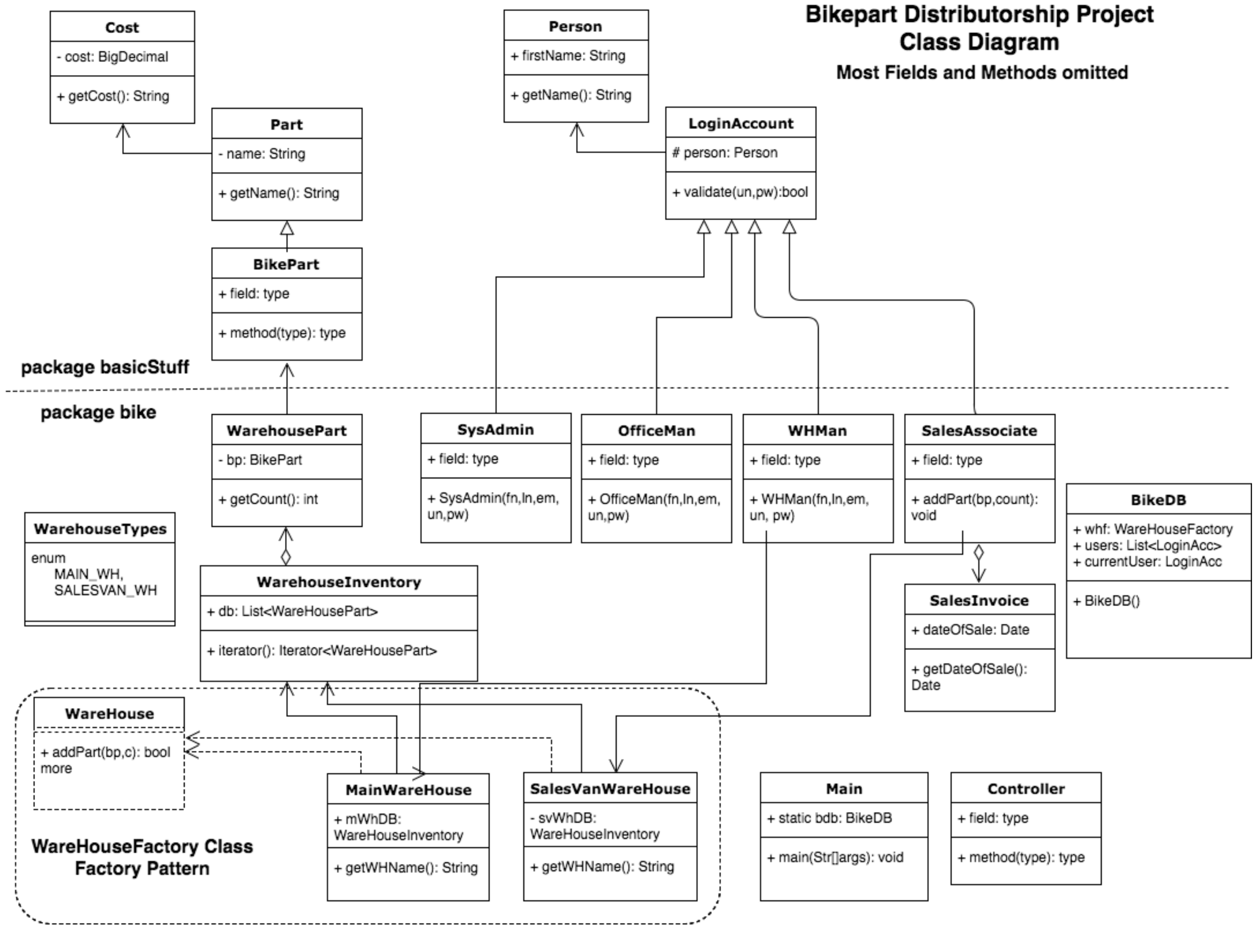
Grades will be assigned based upon all aspects of the project – group progress check, GitHub contributions, code, design, presentation, individual reflection, and peer review.

If you do not contribute substantive code to this project (as demonstrated by your own reflection as well as the peer reviews of your group), you earn a grade of zero on this assignment.

Rubric

Criteria	Points	Score
Project Compiles - Having a compiling project does not earn points, but if your project does not compile, you earn a zero on the project.	0	
Code is submitted as a zip file – If you do not submit your code as a zip file, the maximum score will be a 75.	-25	
Presentation and Demonstration (aesthetics of)	10	
Group Progress Check	10	
GitHub Contributions	10	
Correctness of Code (as Demonstrated)	30	
Satisfying Object-oriented requirements	20	
Individual reflection	10	
Peer review	10	

UML Class Diagram of One Solution



Writings Overview

CPSC 240 satisfies a UMW writing intensive requirement. We have three writing assignments

Writing Assignments

- Writing1IDEUsersGuide.docx
- Writing2BookReport.docx
- Writing3ProjectSpecification.docx

Writing Assignment 1 – User's Guide for an IDE

Learning Objectives

- Gain experience in technical writing by creating a software user's guide
- Learn to verbalize technical concepts for a less technical audience
- Familiarize yourself with a Java Integrated Development Environment (IDE) and connecting the IDE to GitHub

Potential IDEs for Use in CPSC 240

There are several Java Integrated Development Environments, ranging from simple (like BlueJ) to medium (like JGrasp) to all-encompassing (like IntelliJ and NetBeans). You can complete this class with an IDE of your choice; however, selecting an IDE that supports the following will help with your success.

- You want an IDE that allows you to do the following.
    - Organize projects into a collection of classes that could be in different packages.
    - Edit code, compile code, and run projects.
    - Test and debug projects.
    - Consult with other users of the IDE via the Internet.
- You want an IDE that supports our Graphical User Interface (GUI) labs and projects that use JavaFX and SceneBuilder.
    - You can run SceneBuilder as a standalone program to generate the .fxml file and integrate the file with your IDE, but an IDE that incorporates support is easier.
- You want an IDE that supports our labs and projects that use GitHub for storing code.
    - You can access GitHub from the command line interface, but an IDE that incorporates support is easier.
- You want an IDE that supports JUnit testing.
    - You can run JUnit test from the command line interface, but an IDE that incorporates support is easier.

The following are four Java IDEs, with my comments about each. All four are free. B12 and B13 are the CPSC labs in the basement of Trinkle.

- IntelliJ Community Edition is available at https://www.jetbrains.com/idea/
    - IntelliJ is a Java IDE.
    - I use IntelliJ. I like IntelliJ.
    - IntelliJ is a full-featured IDE with a large support community, lots of documentation, and it is easy to use.
    - IntelliJ supports JavaFX, SceneBuilder, GitHub, and JUnit.
    - IntelliJ is installed B12.
- NetBeans is available http://www.oracle.com/
    - NetBeans is a Java IDE.
    - I use NetBeans. I like NetBeans.
    - NetBeans is a full-featured IDE with a large support community, lots of documentation, and it is easy to use.
    - NetBeans supports JavaFX, SceneBuilder, GitHub, and JUnit.
    - NetBeans is installed in B12 and B13.

- Eclipse is available at https://eclipse.org/
  - Eclipse is an IDE that supports many languages. Eclipse supports Java.
  - I have not used Eclipse. Many people use Eclipse.
  - Eclipse is a full-featured IDE with a large support community and lots of documentation.
  - Eclipse supports JavaFX, SceneBuilder, GitHub, and JUnit.
  - Eclipse installed in B12 and B13.
- JGrasp is available at www.jgrasp.org
  - JGrasp is a Java IDE.
  - I have not used JGrasp. Some CPSC 220 classes use JGrasp.
  - JGrasp seems to be medium-featured IDE with a large support community and lots of documentation.
  - I do not know if JGrasp supports JavaFX, SceneBuilder, GitHub, and JUnit.
  - JGrasp is installed in B12.

IDE for this Writing Assignment

You can select any IDE for this writing assignment, but I recommend you select one that you will use to develop code in CPSC 240. This allows this project to serve as a learning springboard for the remainder of the course. You should do some initial research to determine if your current IDE supports all features in the previous section.

Task

Imagine that you are a member of the programming team responsible for developing software with your selected IDE. Your team has recently completed the development and testing of a software project using your selected IDE. At the weekly team meeting, your supervisor assigns you the task of writing a user's guide for the IDE so that new employees can learn how to use the IDE. Your supervisor says the guide shall provide a brief overview of the process to create a Java source file, compile it, and execute it within the IDE. Following the brief overview, the user's guide will describe using the IDE with a local Git repository and GitHub. This description will include establishing the local Git repository, committing changes to the local Git repository, connecting the project to GitHub, and pushing/pulling to/from GitHub. Your supervisor has specified that the user's guide should be hands-on and include activities for the reader to complete to assist him/her in understanding how to use the IDE.

Audience

Any time that you're writing a technical document, you must consider the audience because (for example), writing a user's guide for novice programmers is different than one for experienced.

Since your user's guide targets new employees, the guide should be understandable to novice computer scientists who are familiar with programming, have used an IDE, but have not used Git and GitHub. Think about how you would explain information to someone taking CPSC 240.

Remember that the reader may be unfamiliar with terminology used in the IDE. Be sure to explain any new terms that you use in the guide. As you write your user's guide, anticipate questions and problems that the reader might have as they use the IDE and provide the reader with guidance and explanations in your document.

Content

The paper should include a minimum of four labeled sections, which can include subsections as needed.

1. An introduction section introducing the IDE application. This section answers questions such as: What is the purpose of the guide? What is the IDE used for? What benefits does this IDE offer?
2. A section with the brief overview of using the IDE to edit, compile, and run a Java program. Assume that the reader already has the IDE software installed on his/her computer. This answers questions such as: What does the environment look like when it comes up? What should the reader expect to see on the screen?
3. A section that describes how to use the IDE with a local Git repository and GitHub. This section contains much of your writing. You should walk the user through the steps required to create a local Git repository, to commit changes to the local Git repository, the connect the IDE to a GitHub repository, to push changes to the GitHub repository, and to pull code from the GitHub repository. In addition to explanatory text, provide hands on exercises (step-by-step instructions) that the user can carry out while reading the guide to enhance his or her understanding.
4. A summary section to conclude the guide and providing information about where the reader can find more support and additional information.

Paper Format

- There is technically no minimum length for the paper. High quality papers in the past have been at least 4 typed single-spaced pages using standard margins and a standard 12-point font.
- Please restrict the length of your guide to 5 single spaced pages (not counting images). If you prefer to format use double spacing or 1.5 spacing, that’s fine, but in these cases, your document should cover additional pages.
- Use labeled section headings. While section headings are part of papers for other courses, they are essential for technical documents like a user’s guides.
- Any example Java code provided with in the text of the guide should be obviously delimited from the rest of the text. You can use a different font, size, or color to help you to demit it from the rest of the paper.
- You should also distinguish menu option names from surrounding text in a similar fashion.
- You are encouraged to include screen-shots or other graphics in your paper, but do not include them in your page count. If you do you figures or graphics, label them in the text with a number and title. Example: Figure 1. IntelliJ Initialization Screen.
- You can refer to the reader as “you” throughout the guide. Try to avoid referring to yourself in the paper at all since you’re writing the guide as a representative of your company.

Before you turn in your paper, ask yourself

1. Does the guide have a clear introduction and summary?
2. Does the introduction arouse the reader's interest and indicate what the guide intends to accomplish?

3. Are examples provided that make it easy for the reader to understand the processes described?
4. Is the tone appropriate and consistent?
5. Does the guide adequately explain technical topics to a beginner programmer?
6. Is the guide logically and syntactically sound both as a whole and in individual paragraphs and sentences?
7. Does the guide provide a step-by-step process that is easy to follow?
8. Does the guide serve its intended purposes?
9. Does the guide have the format described in the writing assignment description?

Deliverable

Turn in your paper on Canvas. Please turn in a Word document (preferred) or PDF.

Evaluation Criteria

The grading rubric is posted on Canvas.

Extra Credit

You can earn 1⁄2 a letter grade of extra credit by visiting the Writing Center with a draft of your paper on or before Thursday, September 7. Please indicate the date that you visited the writing center as a comment when you turn in your paper on Canvas. I will receive documentation from the Writing Center to verify your visit.

User's Guide Grading Criteria

Criteria	**Ratings**	**Pts**
Introduction is clear, arouses the reader's interest, and indicates the guide's purpose		12
Guide provides steps that are easy to follow		13
Guide contains examples that make it easy to follow processes		13
Guide explains topics to a novice user		13
There is a clear summary section		12
Uses a format consistent with the assignment description		12
Uses appropriate and consistent tone		12
Guide is logically and syntactically sound		13
		100

Writing Assignment 2 – Book Report

In life and computer science, there is always a new skill to learn or a new perspective to consider. Reading is one of my favorite ways to learn. During the semester, you must read and report on one book from the list below. Several are available from Simpson library. Others are available used from your favorite book retailer for < $25. I show the prices on Amazon.

Step 1 – Select a Book from the Following

Nine Algorithms that Changed the Future
John MacCormick, 2013, 210 pages. ($13.00 Amazon)

Tubes: A Journey to the Center of the Internet
Andrew Blum, 2013, 263 pages, ($9.40 Amazon)

Stumbling on Happiness
Daniel Gilbert, 2007, 259 pages, ($9.98 Amazon)

The Mythical Man-Month: Essays on Software
Frederick P Brooks, 1995, 336 pages, (available online and in print at Simpson Library)

The Passionate Programmer: Creating a Remarkable Career in Software Development
Chad Fowler, 2009, 232 pages, ($12.10 Amazon)

Moonwalking with Einstein
Joshua Foer, 2012, 298 pages, ($8.17 Amazon)

The Agile Samurai
Jonathan Rasmusson, 2010, 264 pages, ($20.62 Amazon)

The Thrilling Adventures of Lovelace and Babbage: The (Mostly) True Story of the First Computer (Pantheon Graphic Novels)
Sydney Padua, 2015, 285 pages, ($18.88 Amazon)

Step 2 – Execute the Assignment Phases

Phase 1 – Select a Book, Due Class 7 (Tuesday, Sep 19)

Submit selected book on Canvas. You should have the book in hand at this point. With your permission, your name and book selection will be posted on the course website. If you don't want your name/book selection to be posted, please indicate.

Phase 2 – Discuss the book, Class 13 (Tuesday, Oct 10)

Meet with other students in the course who are reading the same book. I will provide a list of students and their selected books. You can discuss the book with students from any section of the course. Discuss themes in the book and what you thought about it. You may find it helpful to meet multiple times (perhaps once early on and once after you've completed the reading). Part of our class on 10/10 is dedicated to book discussion.

Phase 3 – Summary of discussion, Due Class 14 (Friday, Oct 13)
Each group member must submit a brief (approximately one-page) summary of your discussion on Canvas. Obviously, each group member's discussion summary will be similar.

Phase 4 – Final Paper, Due Class 15 (Thursday, Oct 19)
Submit your report on Canvas.

Book Report Content
In your book report answer these questions:
- What were the key points of the book?
- Who is the intended audience of the book? (Who would benefit from reading?)
- What lessons did you learn that relate to the field of computer science?
- How can you apply the points to your own situation in life, as a student of computer science, or in your future career?

You can find more guidance on how to structure a book report/review by consulting these guides at the Purdue Online Writing Lab:
- http://owl.english.purdue.edu/owl/resource/703/1/
- http://owl.english.purdue.edu/owl/resource/704/1/

Your report should be approximately 3-4 pages, double-spaced. Write this paper on your own. It's not a group effort.

Additional Tips
- If you quote or use material from the book that you read, you do not need to cite it. In this style of book report/review, it's assumed that any quotes are drawn from the book. If you use material from other books/sources, cite those using any standard citation style (APA, MLA, Chicago, etc). The Purdue Online Writing Lab provides information on citation styles at https://owl.english.purdue.edu/owl/resource/949/1/.
- Be sure to include information about how the book applies to you. Reflection on the book's content is a key component of this assignment.

Book Report Rubric
Criteria	**Ratings**	**Pts**
Strong introduction		10
Summary of book		35
Reflection & application to self		25
Conclusion		10
Writing Style/Grammar/Spelling		20
		100

Writing Assignment 3 – Project Specification Document

Project 3, our group project, has existing descriptive words. Your group must create its own project 3 specification document. The Project 3 description can be improved. You must read the Project 3 description, form your own opinions/assumptions, think about the design together, and determine which of your assumptions are incorrect (or perhaps another stakeholder may need to clarify). At the end of this phase, the requirements should be spelled out in sufficient detail such that there is very little left unsaid about *what* the system will do and how you will accomplish it.

Your document should be written for the stakeholders of the system. A wide audience, including developers, clients (people requesting the system), and end users, may read it. Try to avoid unnecessary technical jargon.

You should provide an overview of the document and your system. Describe in detail the concept of the system that you are building. Document the functionality that the system will have by including use case descriptions and diagrams. Document your system design.

Important Notes

Note 1

Describing a feature is a heck of a lot quicker than designing it, which in turn is a heck of a lot quicker than coding it and testing it. Therefore, you will be able to describe far more features than you will have time to implement in this course.

For this reason, please understand that describing a feature in your use case model does <u>not</u> obligate you to implement that feature in code later. You should indicate the priority of features – for example essential priority features will be implemented, medium priority features are planned to be implemented, and low priority features will be implemented if time permits.

Each team member should author at least four use cases. If you're planning on building a pretty bare bones Bike Parts Distributorship system, your final implementation might not even have twelve use cases. That's okay. Imagine sufficient functionality in this phase to have a good twelve or more.

Note 2

Note that as a team, you should review each other's work and give feedback. This is the only way that a truly quality result will emerge. A big part of the goal of creating this document is to ensure that the whole team has a consistent understanding of the direction of your project.

Note 3

Remember the goal. The goal is not to do a bunch of busy work, but to elucidate the system behavior so that your team is all on the same page when you begin design. To be honest, the deliverables themselves are just the icing on the cake – the cake itself is the process of exploring, brainstorming, and negotiating so that you emerge from this phase with a thorough understanding of what it is you want to build. The deliverables are simply the tangible record of that understanding.

Stuff always changes, and as you begin design and implementation in later phases, you will inevitably change your mind about things and encounter issues that you didn't think of before. That's okay. That's part of software development. You can go back and refine your use cases in later phases as you discover these issues and confront the reality of change.

Think ahead. The use cases you write in phase 1 will have a profound effect on what happens later. You want to resolve ambiguity about what the system is going to do, so that you have a joint understanding moving into the implementation phase.

Note 4

In addition to including diagrams, explain everything in text form.

Note 5

Don't forget to number and label all figures that you include.

Specification Document Template

Title Page with Index

1. Introduction

- Provides an overview, explains the document purpose, and how it is organized.

1.1. Document Purpose

- Explain the purpose of this specification document.
- Make sure that you identify the intended audience for the document.

1.2. System Scope

- Briefly describe the system that you're building. Provide the motivation for this system.
- Include a very high level overview (1-2 paragraphs).

1.3. Overview of the remainder of the document

- Explain how your document is organized. Write a sentence or two summarizing each large section so that if the reader wanted to know where to find some information about the system, they'd have a good idea of where to look within this document.

2. Overall Description

- This section should discuss background information about your system including a definition of client/s, definition of users, and detailed description of the included functionality.   This information should be broken into subsections. One example breakdown is shown below.

2.1. Client characteristics

- Who is your client? Why is this system important to him/her/them? This is fictional and can be made-up.

2.2. User characteristics

- Who are the intended users of the system?   Describe them. What characteristics of your users make them unique? How do users interact with your system?

2.3. Product functions

- Here's where you should describe the software in more detail than you did in section 1.2 (probably about a page overview plus your use cases).

- What type of functionality will be included? Describe the features in detail.
- Document the system functionality with a set of written use case and a system-level use case diagram.

3. System Specification
Explain your system design. What data will be stored in each class? What operations will each class include? What type of relationships will exist between classes in your system? How will the various classes interact? In addition to your text explanations, include a class diagram for your system.

4. Assumptions
List any assumptions that you have made about the system. You should have a brief introduction statement/paragraph in this section. The actual assumptions should be communicated as a bulleted list.

What to Submit, Due Class 21 (Friday, Nov 10)

- Submit on Canvas one specification document for your group.

Completing the specification document should be priority number one this week. As a group, you need to have a consistent vision for the system, how it will be used, what it includes, what it omits, and how it is organized. It is best to suspend implementation until the project is understood at the specification level.

Specification Rubric

Criteria	**Ratings**	**Pts**
Uses the Template: Includes all the required sections. Numbers the sections for easy reference and organization.		5
Introduction: Provides a description of the document's purpose, a brief overview of the system, and a description about how the document is organized.		10
Description: Includes a description of the system, the client, and the end users.		20
Functionality w/Use Case Diagram: Includes a use case diagram and 12-16 numbered use case write-ups.		20
Specification & Class Diagram: Describes the system design in text and includes a system level class diagram		20
Assumptions: Bulleted list of assumptions made. Don't forget to include a sentence or two introducing the list.		5
Tone/Writing Style: Sounds professional and uses a consistent format		10
Grammar & Spelling		10
Total		100