Final Exam Review

CS 4354
Summer II 2015

Jill Seaman

Final Exam

- Thursday, August 6, 11AM-1:30PM
- Closed book, closed notes, clean desk
- Content (Comprehensive):
  - ✦Textbook: Chapters 2, 4+5, 8
  - ✦Java Lectures, GRASP, JUnit + Refactoring
- Weighted more towards content from second half of the course
- 35% of your final grade
- I recommend using a pencil (and eraser)
- I will bring extra paper and stapler, in case they are needed.

Exam Format

- 100 points total
  - ✦Multiple choice questions
  - ✦Drawing UML diagrams
  - ✦Writing programs/functions/code
  - ✦Tracing code (what is the output)
  - ✦Reading diagrams and code (what does it mean, is it a good design)
  - ✦Short answer: “List and describe…”
- Each question will indicate how many points it is worth

Java: Introduction

- Compilation, execution (byte code)
- Features
  - ✦Object-oriented, inheritance, polymorphism, garbage collection
  - ✦Exception handling, concurrency, Persistence, platform independence
- Objects are references (pointers)
- Types:
  - ✦Primitive types
  - ✦arrays, ArrayList
  - ✦classes, methods
- Java library, API
  - ✦import statement

Java: Introduction

- Javadoc, how to document the elements of a program
- Operators, assignment, control flow
    - ✦Similar to C++
- String, toString
- Constructors, this
- Packages
- static, final
- public, private, protected, [package]
- Collections, Maps, Iterators
    - ✦Know how to write code with these (as in the assignments)

Java: Input/Output

- Input using a Scanner
- Output using System.out.println()
- Formatting using the DecimalFormatter or other method

- Object serialization
    - ✦ObjectInputStream, ObjectOutputStream
    - ✦readObject, writeObject
    - ✦Understand how it works.

Java: Inheritance

- Reuse: Composition and Inheritance
- Inheritance
    - ✦class hierarchy: superclass, subclass, (extends keyword)
    - ✦overriding methods, upcasting, constructors
- Polymorphism
    - ✦upcasting, polymorphic functions, dynamic binding, extensibility
- Abstract methods and classes
- Interfaces
    - ✦Implementing interfaces, Multiple inheritance
    - ✦Sorting: implementing Comparable
    - ✦Extending an interface

Java: Exceptions and Threads

- Exceptions
    - ✦Semantics (how exceptions are thrown/caught), syntax
    - ✦Catch or specify requirement (how to satisfy)
    - ✦Runtime exceptions
    - ✦Create your own exception classes (and instances)
- Threads
    - ✦Thread class, Runnable interface
    - ✦Using the above to implement multi-threading
    - ✦Thread methods, what they do.

Ch 2: Modeling with UML: UML diagrams

- Use Case Diagrams
  - ✦Actors, Goals, relationships: inclusion, extension, inheritance
- Class Diagrams
  - ✦Classes, attributes, operations, objects, links/associations
  - ✦unidirectional, bidirectional associations, roles, multiplicity
  - ✦Aggregation, composition, inheritance
- Sequence Diagrams
  - ✦Object interactions, Objects along top, Labels on arrows indicate messages.
- Activity Diagrams
  - ✦Activities, control flow, decisions, forks and joins, swimlanes
- State Machine Diagrams
  - ✦State is a node, event is a directed edge labeled: Event[Guard] / Action

Ch 4-5: OO Software Development: Requirements elicitation and analysis

- Requirements Elicitation
  - ✦Activities: Identifying actors, scenarios, use cases, relationships
- Analysis Activities (from use cases to objects)
  - ✦Identifying Entity Objects, (Boundary Objects), Control Objects
  - ✦Mapping Use Cases to Objects with Sequence Diagrams
  - ✦Identifying Associations, Aggregations, Attributes
  - ✦Modeling Inheritance Relationships
  - ✦Modeling State-Dependent Behavior of Individual Objects
- See Assignment 4

GRASP: Assigning Responsibilities to Objects

- GRASP
  - ✦Deciding which classes should perform which operations
  - ✦Information Expert: That which has the information does the work
  - ✦Creator: Assign class B the responsibility to create instances of class A if:
    - B aggregates A objects.
    - B contains A objects.
    - B records instances of A objects. OR
    - B has the initializing data that will be passed to A
  - ✦Low Coupling: Keep the amount of dependency on other classes low.
  - ✦High Cohesion: Keep the tasks of a class focused and related to each other.
  - ✦Controller: Use a controller class to separate the UI from the entity objects.
- See Assignment 4

Ch 8: Object design: Reusing pattern solutions

- Concepts
  - ✦Specification Inheritance vs Implementation Inheritance, Delegation
- Design Patterns

✦Adapter Pattern	✦Command Pattern
✦Strategy Pattern	✦Composite Pattern
✦Observer Pattern	✦Proxy Pattern
✦Abstract Factory Pattern	✦Facade Pattern

- Framework vs Class Library vs Design Pattern vs Component
- See Assignment 5

JUnit

- JUnit
  - ✦Open source framework for writing and running unit tests
  - ✦Provides automation
  - ✦Annotations: @Test, @Before, @After, @Ignore
  - ✦Assert Methods: fail, assertTrue, assertFalse, assertEquals, assertNull
  - ✦Separation of testing code from production code
  - ✦Testing methods, classes, and collaborating classes.
  - ✦Be able to write simple test cases, using the Annotations and Assert methods as we did in Assignment 6.

Refactoring

- Refactoring
  - ✦Disciplined technique for changing a software system: altering its internal structure without changing its external behavior Provides automation
  - ✦What are the benefits, when is it applied?
  - ✦Know some various refactorings
  - ✦Know the “bad smells” and how to fix them.
  - ✦Be able to apply simple refactorings “by hand”
  - ✦See Assignment 7 (the lab)

Sample Questions: Multiple Choice

- You are designing a datatype to store a mathematical expression composed of binary operations (plus, minus, times, divide) and numbers, such as: (3+4)/(6*2). These expressions can be drawn as a tree with the operations as the node values and the numbers as the leaves. What design pattern should you use for this?

  (a) adaptor (b) composite (c) facade (d) proxy

- Which of the following is NOT potentially an example of refactoring :
  - (a) Push down field.
  - (b) Moving a class to a different package.
  - (c) Adding a new feature requested by a customer.
  - (d) Making your code implement the Facade pattern (without changing the external behavior of the system).

Sample Questions: Analysis & Design

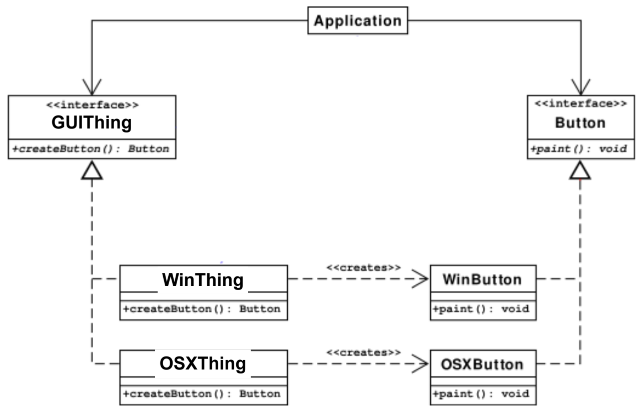
- Given this use case for reserving airline tickets:

Use case name	Book Flight
Entry condition	The Customer is logged in to the system.
Flow of events	1. Customer enters departure and arrival date 2. Customer enters start and end destination 3. System displays available flights (after searching in the flight database): departure date, time, and airport, arrival date, time, and airport, and the airline and price. 4. Customer chooses a particular flight 5. System reserves tickets by adding them to the Customer’s account, but marks them as unpaid.
Exit condition	The Passenger has the selected ticket.

- Draw a **class diagram** showing the structure of data involved in the use case. Include attributes, associations, multiplicity, and operations.
- Draw a **sequence diagram** of the operations involved in the use case.

Sample Questions: Design Patterns

- What design pattern is this an example of?



Sample Questions: Java programming

- Draw a class diagram showing the structure of data about employees of a given company. The employees attributes include name, street address, city, state, zip, and an id number. Full-time employees have an annual salary. Part time employees have an hourly pay rate. Departments have names and a group of employees, but each employee can be in only one department. Employees work on one or more projects, which also have names. Projects may have multiple employees assigned to them.
- Implement in Java the Employee class structure described above. The Employee class should have a polymorphic function called weeklyPay. For Full time employees, their weekly pay is their salary divided by 52. For part time employees their salary is their hourly pay rate time 40. In another class called Driver, define a main function that creates an array or ArrayList of Employees of two full time and one part time employees, then iterates over the list and outputs the name and weekly pay for each employee.