

Objectives

- Designing APIs
- Reviewing the semester

Dec 7, 2011

Sprenkle - CSCI209

1

Represent Thanksgiving?

```
dinner = new Turkey( new Duck( new Chicken() ) );
```

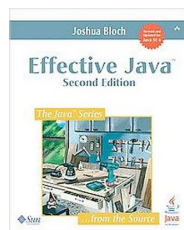
Dec 7, 2011

Sprenkle - CSCI209

2

Josh Bloch's

DESIGNING A GOOD API



Dec 7, 2011

Sprenkle - CSCI209

3

Motivating Good API Design

- APIs can be among a company's greatest assets
 - Customers invest heavily: buying, writing, learning
 - Cost to stop using an API can be prohibitive
 - Successful public APIs capture customers
- Can also be among company's greatest liabilities
 - Bad APIs result in unending stream of support calls
- Public APIs are forever - one chance to get right

How does API design relate to this class?

Dec 7, 2011

Sprenkle - CSCI209

4

How Does API Design Relate To This Class?

- You are an API designer
 - Good code is modular—each module has an API
 - Your teammates will need to use your API
- Useful modules tend to get reused
 - Once module has users, can't change API at will
 - Good reusable modules are corporate assets
- Thinking in terms of APIs improves code quality

Dec 7, 2011

Sprenkle - CSCI209

5

Discussion

- What are examples of APIs we have used?
 - Are they examples of good or bad APIs?
- What are some characteristics of the good? The bad?
 - Make into characteristics of good APIs

Dec 7, 2011

Sprenkle - CSCI209

6

Characteristics of a Good API

- Easy to learn
- Easy to use, even without documentation
- Hard to misuse *What are some techniques to achieve this goal?*
- Easy to read and maintain code that uses it
- Sufficiently powerful to satisfy requirements
- Easy to extend
- Appropriate to audience

Dec 7, 2011

Sprenkle - CSCI209

7

API DESIGN PROCESS

Dec 7, 2011

Sprenkle - CSCI209

8

Discussion

- How to design a good API?
- What is the process?

Dec 7, 2011

Sprenkle - CSCI209

9

API Design Process

1. Gather requirements
 - Eye towards generality
 - Know what is actually required
 - Form: Use cases
2. Write short specification
3. Write to API early
4. Maintain realistic expectations

Dec 7, 2011

Sprenkle - CSCI209

10

2. Write Short Specification

- Goal: 1-page Specification
 - Agility trumps completeness
 - Easier to modify
- Ask as many programmers as possible about spec
- Flesh it out as you gain confidence
 - Necessarily involves coding

Dec 7, 2011

Sprenkle - CSCI209

11

3. Write to Your API Early and Often

- Before you've implemented API, write the code (use cases) that will use the API
 - Saves you doing implementation you'll throw away
 - Similar to role playing
- Start before you've even specified it properly
 - Saves you from writing specs you'll throw away
- Continue writing to API as you flesh it out
 - Prevents nasty surprises
 - Code lives on as **examples, unit tests**

Dec 7, 2011

Sprenkle - CSCI209

12

4. Maintain Realistic Expectations

- Most API designs are over-constrained
 - People want them to do a lot more
 - You won't be able to please everyone
 - Aim to displease everyone equally
- Expect to make mistakes
 - A few years of real-world use will flush them out
 - Expect to evolve API

Dec 7, 2011

Sprenkle - CSCI209

13

To See More

- Slides:
 - <http://icsd05.cs.tamu.edu/slides/keynote.pdf>
- Presentation:
 - <http://www.youtube.com/watch?v=aAb7hSCtvGw>

Dec 7, 2011

Sprenkle - CSCI209

14

Bin Fitting Problem

ASSIGNMENT 9

Dec 7, 2011

Sprenkle - CSCI209

15

Assignment Review

- What is the overall goal?
- What are the responsibilities?
- Thoughts on refactoring/implementation in retrospect?

Total of less than 150 lines of code

Dec 7, 2011

Sprenkle - CSCI209

16

Compare the Following Methods

- `PriorityQueue<Disk> fitAlg(List<Integer> data, PriorityQueue<Disk> bins)`
- `PackingResult fitAlg(List<Integer> fileList)`
- `int fitAlg()`

How is testing affected?

Dec 7, 2011

Sprenkle - CSCI209

17

API Discussion

- What is a good comment for this method?

```
public static void printResults(PriorityQueue<Disk> bins,
double totalsize, String method) {
    System.out.println("-----");
    System.out.println(method);
    System.out.println("total size = " + totalsize + "GB");
    System.out.println();
    System.out.println("number of bins used: " +
        bins.size());
    System.out.println("Bin# Free Space File sizes");
    while (!bins.isEmpty()) {
        System.out.println(bins.poll());
    }
    System.out.println();
}
```

Side Effect!

Dec 7, 2011

Sprenkle - CSCI209

18

Excerpts from Good Critiques

- Unfortunately, because of the linear style of programming used by the original author, we can't debug this program quickly or efficiently. Debugging would basically consist of checking to make sure that each section of the code works as intended, which means we'd have to test the code basically by every 10 lines or so....

Dec 7, 2011

Sprenkle - CSCI209

19

Excerpts from Good Critiques

- Added a static field "ID" to track the ID of a disk rather than wasting the extra code lines of having an extra constructor to specify the ID and forcing [others to] track the IDs of the disks it is creating...
- The downside of this approach is that we can't directly specify what we want the ID of a disk to be. On the other hand, it is a much more direct and efficient way to ensure that we are always getting a unique set of IDs for a set of disks.

Dec 7, 2011

Sprenkle - CSCI209

20

Excerpts from Good Critiques

- The majority of the bin-fitting process was handled inside the main method. This probably made the code easy to write, but is disadvantageous for a number of reasons:
 - Readability: ...
 - Maintainability: ...
 - Testing: *unit testing does not break down into small pieces to test*. There is just one big main method
 - Debugging: ...
- The bins class was not object oriented really...

Dec 7, 2011

Sprenkle - CSCI209

21

Excerpts from Good Code Critiques

- After looking back over the code and the changes I've made, I think there will almost always be more changes possible. For example, the code for the different heuristic types could be extracted to a separate class that's [sic] only job is to define the heuristics.
- Also, the Disk class could be changed to accommodate any type of storage media, not just DVDs.

Dec 7, 2011

Sprenkle - CSCI209

22

Excerpts from Good Critiques

- I thought about how this program *is likely to change*. Right now we have two different methods to fit files onto disks; however, these two are certainly not the only two methods, and in the future *perhaps we will want to use other methods* in the Bins class. For this reason, I decided to make the fitFilesToDisk method abstract in the Bins class and to make a WorstFit class that inherits from the Bin class....

Dec 7, 2011

Sprenkle - CSCI209

23

ASSIGNMENT 10 DISCUSSION

Dec 7, 2011

Sprenkle - CSCI209

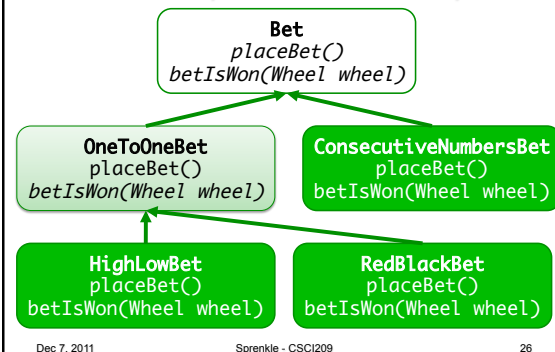
24

Review: Game class

```
private String placeBet(int whichBet) {
    String result = "";

    if (whichBet == 0) {
        Set<String> choices = new TreeSet<String>();
        choices.add(Wheel.BLACK);
        choices.add(Wheel.RED);
        result = ConsoleReader.promptOneOf("Please bet",
            choices);
    } else if (whichBet == 1) {
        Set<String> choices = new TreeSet<String>();
        choices.add("even");
        choices.add("odd");
        result = ConsoleReader.promptOneOf("Please bet",
            choices);
    } else if (whichBet == 2) {
        ...
    }
    System.out.println();
    return result;
}
```

One Possibility for Bet Hierarchy



Discussion

- Benefits of the refactored hierarchy
- Drawbacks of the refactored hierarchy

Dec 7, 2011

Sprenkle - CSCI209

27

Benefits of The Refactored Hierarchy

- Benefits of the refactored hierarchy
 - Where is the logic about the bets?
 - In the Bet classes
 - Game can manage the game, not be responsible for bets
 - Easier to add a new Bet
- Drawbacks of the refactored hierarchy
 - Adds more classes, hierarchy, abstraction

Dec 7, 2011

Sprenkle - CSCI209

28

What have you learned this semester?
What are you taking with you?

OH, THE PLACES YOU HAVE BEEN!

Dec 7, 2011

Sprenkle - CSCI209

29

CSCI209 Objectives

- Talk about software development and practices knowledgeably, using appropriate **terminology**
- Design, implement, test, and document efficient applications of increasing size and complexity
- Understand designs and implementations of others
- Use a version control system
- Use many of the capabilities of the Eclipse IDE
- Test and debug large applications systematically, using standard tools
- Understand design principles
- Discuss benefits and limitations of a statically-typed language

Dec 7, 2011

Sprenkle - CSCI209

30

My Philosophy

- Balance imparting knowledge and creating learning experiences
- Goals
 - Help you recognize bad design, fixes for it
 - Learn to read others' code—not just mine
 - Transferable skills
 - VCS, IDE use, abstraction, design
 - Best practices of Java
 - Small assignments on Java specifics
 - *Effective Java*

Dec 7, 2011

Sprenkle - CSCI209

31

Summary of Java Platform SE 6.0

Remember from the first day of class?

Java Language	Java Language									
Tools & Tool APIs	Java	javac	javadoc	apt	jar	javap	JPDA	jconsole		
	Security	Int'l	RMI	IDL	Deploy	Monitoring	Troubleshoot	Scripting	JVM TI	
Deployment Technologies	Deployment			Java Web Start				Java Plug-in		
User Interface Toolkits	AWT			Swing			Java 2D			
	Accessibility		Drag n Drop	Input Methods		Image I/O	Print Service		Sound	
Integration Libraries	IDL	JDBC™	JNDI™		RMI	RMI-IIOP		Scripting		
Other Base Libraries	Beans	Intl Support		I/O	JMX	JNI		Math		
	Networking	Override Mechanism		Security	Serialization	Extension Mechanism		XML JAXP		
lang and util Base Libraries	lang and util	Collections		Concurrency Utilities		JAR	Logging		Management	
	Preferences API	Ref Objects	Reflection		Regular Expressions		Versioning	Zip	Instrument	
Java Virtual Machine	Java Hotspot™ Client VM					Java Hotspot™ Server VM				
Platforms	Solaris™		Linux			Windows			Other	

Image from Oracle's site

Dec 7, 2011

Sprenkle - CSCI209

32

Summary of Java Platform SE 6.0

Remember from the first day of class?

Java Language	Java Language									
Tools & Tool APIs	java	javac	javadoc	apt	jar	javap	JPDA	jconsole		
Deployment Technologies	Security	Int'l	RMI	IDL	Deploy	Monitoring	Troubleshoot	Scripting	JVM TI	
User Interface Toolkits	Deployment				Java Web Start			Java Plug-in		
Integration Libraries	AWT				Swing			Java 2D		
Other Base Libraries	Accessibility	Drag n Drop	Input Methods		Image I/O	Print Service		Sound		
lang and util Base Libraries	IDL	JDBC™	JNDI™		RMI	RMI-HOP		Scripting		
Java Virtual Machine	Beans	Intl Support	I/O		JMX	JNI		Math		
Platforms	Networking	Override Mechanism	Security	Serialization	Extension Mechanism		XML JAXP			
	lang and util	Collections	Concurrency Utilities		JAR	Logging		Management		
	Preferences API	Ref Objects	Reflection		Regular Expressions	Versioning		Zip Instrument		
	Java Hotspot™ Client VM				Java Hotspot™ Server VM					
	Solaris™		Linux		Windows			Other		

Dec 7, 2011

Sprenkle - CSCI209

33

Where Will You Go From Here?

- What do you think you're most likely to take with you?
- What will be your design philosophy?

Dec 7, 2011

Sprenkle - CSCI209

34

Project Notes

- Statistics usage
 - How not used
- Project Analysis:
 - Understand the others' design/code/parts
 - At least at a high level
 - Contents: Description, Planning, Status, Code Analysis, Collaboration, Future Work
 - Complete specification online
- Possibility of Friday demos?

Dec 7, 2011

Sprenkle - CSCI209

35

Course Evaluations

- Due Sunday @ midnight
- Two evaluations
 - Standard for Computer Science dept
 - Supplemental, specific to CSCI209
- Incentive to fill out evaluations
 - If 63% fill out, 1% EC on "Individual programming and written homework assignments"
 - Additional 1% for every additional ~9% (each student) who complete
 - 1335 total points possible

Dec 7, 2011

Sprenkle - CSCI209

36