Principles of Software Construction: Objects, Design, and Concurrency

Part 1: Designing Classes

Design for change

**Charlie Garrod** 

**Michael Hilton** 

School of
Computer Science

This image cannot currently be displayed.



# Smoking Section



### **Announcements**

- HW1 is available
- You should have a course repo by now (if you don't, get it soon!)
- HW1 is due Thursday, Sept 7 at 11:59 p.m.
- Office Hours start today!



### Random Calling

- A way to include EVERYONE in the class
- Goal is participation, not "correctness"
- Wrong Answers & Mistakes are Expected and Valued
- We are all part of a learning community
- It helps me learn your names



# I. PolymorphismII.Information hidingIII.Contracts



### Objects

- An object is a bundle of state and behavior
- State the data contained in the object
  - In Java, these are the fields of the object
- Behavior the actions supported by the object
  - In Java, these are called methods
  - Method is just OO-speak for function
  - invoke a method = call a function

IST institute for SOFTWARE RESEARCH

### Classes

- Every object has a class
  - A class defines methods and fields
  - Methods and fields collectively known as members
- Class defines both type and implementation
  - type ≈ where the object can be used
  - implementation ≈ how the object does things
- Loosely speaking, the methods of a class are its
   Application Programming Interface (API)
  - Defines how users interact with instances







- Dice Game
  - -Roll 2 dice
  - —If total value ==7 you win!
  - –Else you lose

## To the computer!



### Class example - Dice



public class Dice { private int faceValue; public void roll() { faceValue = 1 + (int)(Math.random() \* 5); public int getFaceValue() { return faceValue;

### Class usage example

```
public class Main {
       public static void main(String[] args) {
           Dice d1 = new Dice();
           Dice d2 = new Dice();
           d1.roll();
           d2.roll();
           if(d1.getFaceValue()+d2.getFaceValue()==7){
               System.out.println("You Win!");
           }else{
               System.out.println("You Lose");
```

When you run this program, it prints?

### Interfaces and implementations

- Multiple implementations of API can coexist
  - Multiple classes can implement the same API
  - They can differ in performance and behavior
- In Java, an API is specified by interface or class
  - Interface provides only an API
  - Class provides an API and an implementation
  - A Class can implement multiple interfaces

IST institute for SOFTWARE RESEARCH





- Dice Game
  - -Add "loaded" dice
  - -Rules of game are unchanged

## To the computer!





15-214

### An interface to go with our class

```
public interface Dice {
    void roll();
    int getFaceValue();
}
```

An interface defines but does not implement API

### Modifying class to use interface

```
public class RandomDice implements Dice {
    private int faceValue;
    @Override
    public void roll() {
        faceValue = 1 + (int)(Math.random() * 5);
    }
    @Override
    public int getFaceValue() {
        return faceValue;
    }
}
```

### Modifying client to use interface

```
public class Main {
    public static void main(String[] args) {
        Dice d1 = new RandomDice();
        Dice d2 = new RandomDice();
        d1.roll();
        d2.roll();
        if(d1.getFaceValue()+d2.getFaceValue()==7){
            System.out.println("You Win!");
        }else{
            System.out.println("You Lose");
        }
    }
}
```

# Interface permits multiple implementations

```
public class LoadedDice implements Dice {
    public void roll() {

    public int getFaceValue() {
        if(Math.random()> .5){
            return 4;
        }
        return 2;
    }
}
```

### Why multiple implementations?

## Different performance

Choose implementation that works best for your use

### Different behavior

- Choose implementation that does what you want
- Behavior must comply with interface spec ("contract")

# Often performance and behavior both vary

- Provides a functionality performance tradeoff
- Example: HashSet, TreeSet

IST institute for SOFTWARE RESEARCH

### Java interfaces and classes

# Organize program program objects types

- Each type offers a specific set of operations
- Objects are otherwise opaque

### Interfaces vs. classes

- Interface: specifies expectations
- Class: delivers on expectations (the implementation)



### Classes as types

- Classes do define types
  - Public class methods usable like interface methods
  - Public fields directly accessible from other classes
- But prefer the use of interfaces
  - Use interface types for variables and parameters unless you know one implementation will suffice
    - Supports change of implementation
    - Prevents dependence on implementation details

```
Set<Criminal> senate = new HashSet<>();  // Do this...
HashSet<Criminal> senate = new HashSet<>();  // Not this
```



### Check your understanding

```
interface Animal {
    void vocalize();
class Dog implements Animal {
    public void vocalize() { System.out.println("Woof!"); }
class Cow implements Animal {
    public void vocalize() { moo(); }
    public void moo() {System.out.println("Moo!"); }
What Happens?
  1. Animal a = new Animal();
    vocalize();
  2. Dog d = \text{new Dog()};
    d.vocalize();
 3. Animal b = new Cow();
    b.vocalize();
 4. b.moo();
```

# Historical note: simulation and the origins of OO programming

- Simula 67 was the first object-oriented language
- Developed by Kristin Nygaard and Ole-Johan Dahl at the Norwegian Computing Center



Dahl and Nygaard at the time of Simula's development

- Developed to support discrete-event simulation
  - Application: operations research, e.g. traffic analysis
  - Extensibility was a key quality attribute for them
  - Code reuse was another

IST institute for SOFTWARE RESEARCH

15-214

# I. PolymorphismII.Information hidingIII.Contracts



### Information hiding

- Single most important factor that distinguishes a well-designed module from a bad one is the degree to which it hides internal data and other implementation details from other modules
- Well-designed code hides all implementation details
  - Cleanly separates API from implementation
  - Modules communicate only through APIs
  - The are oblivious to each others' inner workings
- Known as information hiding or encapsulation
- Fundamental tenet of software design [Parnas, '72]



### Benefits of information hiding

- Decouples the classes that comprise a system
  - Allows them to be developed, tested, optimized, used, understood, and modified in isolation
- Speeds up system development
  - Classes can be developed in parallel
- Eases burden of maintenance
  - Classes can be understood more quickly and debugged with little fear of harming other modules
- Enables effective performance tuning
  - "Hot" classes can be optimized in isolation
- Increases software reuse
  - Loosely-coupled classes often prove useful in other contexts

IST institute for SOFTWARE RESEARCH

### Information hiding with interfaces

- Declare variables using interface type
- Client can use only interface methods
- Fields not accessible from client code
- But this only takes us so far
  - Client can access non-interface members directly
  - In essence, it's voluntary information hiding



# Mandatory Information hiding visibility modifiers for members

- private Accessible only from declaring class
- package-private Accessible from any class in the package where it is declared
  - Technically known as default access
  - You get this if no access modifier is specified
- protected Accessible from subclasses of declaring class (and within package)
- public Accessible from anywhere

IST institute for SOFTWARE RESEARCH

### Discussion

- You know the benefits of private fields
- What are the benefits of private methods?

### Best practices for information hiding

- Carefully design your API
- Provide only functionality required by clients
  - -All other members should be private
- You can always make a private member public later without breaking clients
  - But not vice-versa!



# I. PolymorphismII.Information hidingIII.Contracts

### Why do we have contracts?

### AGREEMENT TO SELL REAL ESTATE

This is a legally binding contract entered into this day RECEIPT IS HEREBY ACKNOWLEDGED OF THE SUM OF: Two Thousand Five Hundred Dollars (\$2.500 Non Refundable) From as a deposit on account of the purchase price of the following described property upon the terms and conditions as stated herein DESCRIPTION OF PROPERTY: That lot, piece, or parcel of land situated in Maricopa County, State of Arizona. Address APN For the PURCHASE PRICE of TERMS AND CONDITIONS OF SALE Buyer warrants that they are purchasing the property for use as an investment and not as a personal Buyer cannot advertise or promote the property prior to close. Possession of said premises will be given to purchaser at the time of closing. This contract is not assignable. Buyer may not sessil the property without express written consent from Taxes, rent and sent securities shall be prosated at the time of closing and paid by the seller. If Seller has the Property in escrow but has not yet closed on it, the parties agree that they will do an assignment of Seller's contact with the current owner, at Seller's request, so that the Buyer's total purchase price remains the same after the assignment fee is considered Seller agrees to deliver premises at time of closing in an as-is condition. In case the property is destroyed, wholly or partially, by fire or other casualty prior to closing, Buyer or Seller shall have option for ten (10) days thereafter of proceeding to remedy the situation or terminating agreement. Said property is being sold and purchased subject to zoning ordinances and regulations; building restrictions; and conditions, sestrictions and easements of Public Record. It is agreed that time is of the essence with respect to all dates specified in the Agreement and any addenda, riders, or amendments thereto. This means that all deadlines are intended to be strict and absolute. If the closing does not occur by the Closing Date, the Agreement is automatically terminated and the Seller shall retain any earnest money deposit as liquidated damages. In the event Buyer requests an extension of the Closing Date or of the deadline for the fulfillment of any contingency, and the Seller agrees to the extension, the Buyer agrees to pay to the Seller a per diem penalty of the greater of \$200,00 or 1/10 of 1% of the purchase price per calendar day towards Seller's carrying costs, through and including the Closing Date specified in the written extension agreement. The per diem amount must be deposited with the Seller at the time any request for extension is made. Seller agrees to deliver a good and make table or insurable owner's title to the property above described free and clear of all encumbrances except as herein set forth. Buyer shall pay for all closing costs and title fees, including pergantion of deed. Purchaser agrees to notify Seller in wining of any defects in title as soon as reasonably possible and if title proves to be not good and marketable or insurable, the seller is to

make title good and madetable or insurable and shall have a reasonable time from notification so to do.



Setup

In Graph g,

"Tom" and "Anne" are not connected.

Who's to blame?

Algorithms.shortestDistance(g, "Tom", "Anne");

> ArrayOutOfBoundsException



Who's to blame?

Algorithms.shortestDistance(g, "Tom", "Anne");

> -1

Who's to blame?

Algorithms.shortestDistance(g, "Tom", "Anne");

> 6

### Who's to blame?

```
class Algorithms {
     * This method finds the
     * shortest distance between to
     * verticies. It returns -1 if
     * the two nodes are not
     * connected. */
    int shortestDistance(...) {...}
```

Who's to blame?

Math.sqrt(-5);

> 0

Who's to blame?

# **Java Documentation**



15-214

### **Contracts**

- Agreement between provider and users of an object
- Includes
  - Interface specification (types)
  - Functionality and correctness expectations
  - Performance expectations
- What the method does, not how it does it
  - Interface (API), not implementation



### **Textual Specification**

### public int read(byte[] b, int off, int len) throws IOException

- Reads up to len bytes of data from the input stream into an array of bytes. An
  attempt is made to read as many as len bytes, but a smaller number may be read.
  The number of bytes actually read is returned as an integer. This method blocks
  until input data is available, end of file is detected, or an exception is thrown.
- If len is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value -1 is returned; otherwise, at least one byte is read and stored into b.
- The first byte read is stored into element b[off], the next one into b[off+1], and so on. The number of bytes read is, at most, equal to len. Let k be the number of bytes actually read; these bytes will be stored in elements b[off] throughb[off+k-1], leaving elements b[off+k] through b[off+len-1] unaffected.
- In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are unaffected.

#### • Throws:

- IOException If the first byte cannot be read for any reason other than end of file, or if the input stream has been closed, or if some other I/O error occurs.
- NullPointerException If b is null.
- IndexOutOfBoundsException If off is negative, len is negative, or len is greater than b.length - off

IST institute for SOFTWARE RESEARCH

15-214

## **Textual Specification**

### public int read(byte[] b, int off, int len) throws IOException

- Reads up to len bytes of data fattempt is made to read as made to
- If len is zero, then no bytes ar attempt to read at least one by end of file, the value -1 is retuinto b.
- The first byte read is stored in on. The number of bytes read bytes actually read; these byte 1], leaving elements b[off+k]

- Specification of return
- Timing behavior (blocks)
- Case-by-case spec
  - len=0 → return 0
  - len>0 && eof → return -1
  - len>0 && !eof → return >0
- Exactly where the data is stored
- What parts of the array are not affected
- In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are unaffected.

#### • Throws:

- IOException If the first byte or if the input stream has beer
- NullPointerException If b is n
- IndexOutOfBoundsException than b.length - off
- Multiple error cases, each with a precondition
- Includes "runtime exceptions" not in throws clause

ISI institute for SOFTWARE RESEARCH

15-214

### **Specifications**

- Contains
  - Functional behavior
  - Erroneous behavior
  - Quality attributes (performance, scalability, security, ...)
- Desirable attributes
  - Complete
    - Does not leave out any desired behavior
  - Minimal
    - Does not require anything that the user does not care about
  - Unambiguous
    - Fully specifies what the system should do in every case the user cares about
  - Consistent
    - Does not have internal contradictions.
  - Testable
    - Feasible to objectively evaluate
  - Correct
    - Represents what the end-user(s) need

institute for SOFTWARE RESEARCH

## **Functional Specification**

- States method's and caller's responsibilities
- Analogy: legal contract
  - If you pay me this amount on this schedule...
  - I will build a with the following detailed specification
  - Some contracts have remedies for nonperformance
- Method contract structure
  - Preconditions: what method requires for correct operation
  - Postconditions: what method establishes on completion
  - Exceptional behavior: what it does if precondition violated
- Defines what it means for impl to be correct



# **Functional Specification**

- What does the implementation have to fulfill if the client States
- violates the precondition? Analog — If yell
  - I wil , actained specification
  - .... acts nave remedies for nonperformance – Som
- Method contract structure
  - Preconditions: what method requires for correct operation
  - Postconditions: what method establishes on completion
  - Exceptional behavior: what it does if precondition violated
- Defines what it means for impl to be correct



# Formal Specifications

```
/*@ requires len >= 0 && array != null && array.length == len;
   ensures \result ==
               (\sum int j; 0 <= j && j < len; array[j]);
int total(int array[], int len);
```

Advantage of formal specifications:

- \* runtime checks (almost) for free
- \* basis for formal verification
- \* assisting automatic analysis tools

JML (Java Modelling Language) as specifications language in Java (inside comments)

Disadvantages?



## Runtime Checking of Specifications with Assertions

```
/*@ requires len >= 0 && array.length == len
  @ ensures \result ==
              (\sum int j; 0 \le j & j \le len; array[j])
  (a)
  @*/
float sum(int array[], int len) {
    assert len >= 0;
    assert array.length == len;
    float sum = 0.0;
    int i = 0;
    while (i < len) {
        sum = sum + array[i]; i = i + 1;
    assert sum ...;
    return sum;
```

java -ea Main

## **Runtime Checking with Exceptions**

```
/*@ requires len \geq 0 \&\& array.length == len
  @ ensures \result ==
  (a)
               (\sum int j; 0 \le j & j \le len; array[j])
  @*/
float sum(int array[], int len) {
    if (len < 0 || array.length != len)
        throw IllegalArgumentException(...);
    float sum = 0.0;
    int i = 0;
    while (i < len) {
        sum = sum + array[i]; i = i + 1;
    return sum;
    assert ...;
```

Check arguments even when assertions are disabled. Good for robust libraries!

# Specifications in the real world

### Javadoc

```
/**
  Returns the element at the specified position of this list.
                                                                    postcondition
 * This method is <i>not</i> guaranteed to run in constant time.
  In some implementations, it may run in time proportional to the
  element position.
 *
  @param index position of element to return; must be non-negative and
 *
                less than the size of this list.
  @return the element at the specified position of this list
  @throws IndexOutOfBoundsException if the index is out of range
                                                                     precondition
           ({@code index < 0 || index >= this.size()})
 */
E get(int index);
```

### <u>java docs</u>

### **EXERCISE:** Write a Specification

### Write

- a type signature,
- a textual specification, and
- a formal specification

for a function **slice(list, from, until)** that returns all values of a list between positions <from> and <until> as a new list

### **Contacts and Interfaces**

- All objects implementing an interface must adhere to the interface's contracts
  - Objects may provide different implementations for the same specification
  - Subtype polymorphism: Client only cares about interface, not about the implementation

p.getX() s.read()

=> Design for Change

## Summary

- Use interfaces to define APIs
- Information hiding is fundamental for good software design
- Software contracts communicate how software should be used

