Managing Complexity

Divide problem into subproblems that
  Can be solved independently
  Can be combined to solve the original problem

How do we know they can be solved independently?
How do we know they can be combined to solve the original problem?

Abstraction

An abstraction is a many-to-one map.

Using Abstractions

When a client uses an abstraction, it should work as the client expects it to no matter which implementation is provided.

Specification

- Tells the client of an abstraction what the client can expect it to do
- Tells the implementer of an abstraction what the implementation must do to satisfy the client
- Contract between client and implementer:
  - Client will only rely on behavior described by specification
  - Implementer will provide an implementation that satisfies the specification
Formality of Specifications

**Informal:** written in a natural language (e.g., English)
- People can disagree on what it means
- Degrees of informality

**Formal:** written in a specification language
- Meaning is defined by specification language (whose meaning is defined precisely, but eventually informally)
- May be analyzed by machines

Note: people (e.g., Wes Weimer) also attempt to develop programs to analyze informal specifications, but it's much harder!

Good Specifications

- **Clear, precise and unambiguous**
  - Clients and implementers will agree on what they mean
- **Complete**
  - Describe the behavior of the abstraction in all situations
- **Declarative**
  - Describe what the abstraction should do, not how it should do it

All specifications in cs2220 should strive for all of these.

Is it even possible for an informal specification to achieve all these?

What do you call people who decide what informal specifications mean?

Example Informal Specification

*Excessive bail shall not be required, nor excessive fines imposed, nor cruel and unusual punishments inflicted.*

8th Amendment

Correct Implementation?

```java
public static boolean violatesEighthAmendment(Punishment p) {
    // EFFECTS: Returns true if p violates the 8th amendment (cruel and unusual punishments).
    return (p.isCruel() && p.isUnusual());
}
```

Or did they mean `p.isCruel() || p.isUnusual()`?

Procedural Specifications

Specification for a procedure describes:
- What its inputs are including their types and meanings
- The mapping between inputs and outputs
- What it can do to the state of the world
Parts of a Procedure Specification

**Header:** name of procedure, types of parameters and return value
  – Java declaration (this is formal)

**Clauses:** (comments in English)

- **REQUIRES**
  - precondition the client must satisfy before calling
- **EFFECTS**
  - postcondition the implementation satisfy at return

Specifications are **Contracts**

Client promise:
- satisfy the precondition in REQUIRES clause

Implementer promise:
- if client satisfies the precondition, when the function returns, the return value and state will satisfy the postcondition.

---

**Specification Contract**

\[ f() \]

- **REQUIRES:** precondition
- **EFFECTS:** postcondition

\[ \text{precondition} \]
\{ f(); \}
\text{If the precondition is true, after we call \( f() \) the postcondition is true.}

\[ \text{postcondition} \]

**Specification Example**

```java
public String bestStock()

// REQUIRES: false
// EFFECTS: Returns the name of the best stock to buy on the NASDAQ tomorrow.
```

Can we implement a procedure that satisfies this specification?

Yes, any implementation will satisfy this specification! If the precondition in the requires clause is not satisfied, the procedure can do anything and still satisfy its specification!

---

**Specification Example**

```java
public String bestStock()

// REQUIRES: true
// EFFECTS: Returns the name of the best stock to buy on the NASDAQ tomorrow.
```

Can we implement a procedure that satisfies this specification?

---

**Preconditions**

The weaker (more easy to make true) the requires clause:
- The more useful a procedure is for clients
- The more difficult it is to implement correctly

**Avoid preconditions** unless there is a really good reason to have one
- Default requires clause is: **REQUIRES true**
- Client doesn’t need to satisfy anything before calling
Specification Example

```java
public static int biggest (int[] a) 
// REQUIRES: true 
// EFFECTS: Returns the value of the biggest element of a.
```

Is this a good specification?

Clear, precise and unambiguous
Complete
Declarative

Bad Use of Preconditions

Bug discovered in Microsoft Outlook that treats messages that start with “begin ” as empty attachments (can be exploited by viruses)

http://support.microsoft.com/kb/260822

To workaround this problem:
- Do not start messages with the word "begin" followed by two spaces.
- Use only one space between the word "begin" and the following data.
- Capitalize the word "begin" so that it reads "Begin."
- Use a different word such as "start" or "commence."

Specification Example

```java
public static int biggest (int[] a) 
// REQUIRES: a has at least one element. 
// EFFECTS: Returns the value of the biggest element of a.
```

Maybe, depends on the client. Its risky...

Specification Example

```java
public static int biggest (int[] a) 
// REQUIRES: true 
// EFFECTS: If a has at least one element, returns the value biggest element of a. Otherwise, returns Integer.MIN_VALUE (smallest int value).
```

Better, but client has to deal with special case now.
Best would probably be to use an exception...

Specification Example

```java
public static int biggest (int[] a) throws NoElementException 
// REQUIRES: true 
// EFFECTS: Scans through each element in a, checking if the value is bigger than the biggest previous value. Returns the value of the biggest element. If the array is empty, throws NoElementException.
```

Is this a good specification?

Clear, precise and unambiguous
Complete
Declarative

Modifies

How does a client know a is the same after `biggest` returns?

```java
public static int biggest (int[] a) throws NoElementException 
// REQUIRES: true 
// EFFECTS: If a has at least one element, returns the value biggest element of a. Otherwise, throws NoElementException.
```

Reading the effects clause should be enough – if biggest modifies anything, it should describe it. But, that’s a lot of work.
**Modifies**

MODIFIES clause: any state **not** listed in the modifies clause **may not be changed** by the procedure.

```java
public static int biggest (int [] a)
// REQUIRES: true
// MODIFIES: nothing
// EFFECTS: If a has at least one element,
// returns the value biggest element of a.
// Otherwise, returns Integer.MIN_VALUE
// (smallest int value).
```

**Modifies Example**

```java
public static int replaceBiggest (int [] a, int [] b)
// REQUIRES: a and b both have at least one element
// MODIFIES: a
// EFFECTS: Replaces the value of the biggest element in a
// with the value of the biggest element in b.
```

**Defaults**

What should it mean when there is no REQUIRES?

**REQUIRES: true**

What should it mean when there is no MODIFIES?

**MODIFIES: nothing**

What should it mean when there is no EFFECTS?

**Meaningless.**

**Returning PS1 / PS2 Partners**

- Blanton, James jb7bq; Smith, David dcs9z; Wallace, Alexander aww8rj
- Borja, Joseph jb4wa; Noh, Brian bkn3yh
- Brown, Jeremy jpb4s; Marion, John jjm6p
- Chen, Jiamin jczkk; Kalish, Michael mk6af
- Dewey-Vogt, Michael mk6Sm; Sparkman, Elisabeth egs5u
- Dilorenzo, Jonathan jdbbaz; Hearn, Charles cmk3eg
- Dollhopf, Niklaus nnm3ey; Oh, Uyn uha6r
- Featherston, Joseph je8S6z; Sun, Yixin y63kz
- Hearn, Charles cmk3eg; Dilorenzo, Jonathan jdbbaz
- Herder, Samuel srh5n; Lopez, Erik eji3tf
- Kalish, Michael mk6af; Chen, Jiamin jczkk
- Lopez, Erik eji3tf; Herder, Samuel srh5n
- Marion, John jjm6p; Brown, Jeremy jpb4s
- Noh, Brian bkn3yh; Borja, Joseph jb4wa
- Oh, Uyn uha6r; Dollhopf, Niklaus nnm3ey
- Smith, David dcs9z; Blanton, James jb7bq; Wallace, Alexander aww8rj
- Sparkman, Elisabeth egs5u; Dewey-Vogt, Michael mk6Sm
- Sun, Yixin y63kz; Featherston, Joseph je8S6z
- Wallace, Alexander aww8rj; Smith, David dcs9z; Blanton, James jb7bq

**Which is better?**

```java
/**
 * EFFECTS: Plays the old song forever.
 */
public static void playOldSong() {
  Player player = new Player();
  while (true) {
    player.play(tune);
  }
}
```

```java
public static void accelerateSong(String tune, int repeats, int tempo, double rate) {
  double currentTempo = tempo;
  for (int i = 0; i < repeats; i++) {
    playOldSongTempo((int)currentTempo);
    currentTempo *= rate;
  }
}
```

Unlike Python, the Java compiler doesn't care how you indent your code. But, you should!
In Eclipse: use **ctrl-I** to indent your code structurally.

**Which is better?**

```java
/**
 * EFFECTS: Plays the old song forever.
 */
P public static void playOldSong() {
  Player player = new Player();
  player.play(tune);
}
```
Finding Java Documentation

Bing/Google: java se 1.6 ArrayList

There are many old specs you will find first without this; some things have changed, so be careful to use the current specs.

Java compiler error messages:

- cannot be resolved to a variable

Java "cannot be resolved to a variable"

Charge

- Find your PS2 partner now
  - If you haven't already finished part I, finish it soon so you can get started together on part II
- PS2 question 6: a lot for you to figure out on your own (but help will be available, and I’ll provide hints when you ask)
  - Keep things simple
  - Design for testability: check your code as you go