CS 3
Introduction to Software Engineering

2: Procedural Abstraction

(Chapter 3)
Announcement

• Problem Set/Lab 1 on web page.

• Due at 4 a.m. next Thursday. ("Wednesday night")
Reading

• Before next time:
  – Read Chapter 1.
  – Skim Chapter 2 (Java notions).
  – Read Chapter 3.

• Before or after next time:
  – Read Chapter 4.

• From now on, readings for each lecture on web page. (www.cs.caltech.edu/courses/cs3/)
Procedural Abstraction

• You write procedures all the time.

• Other than “The lab handout said I should,” why do you define a new procedure?
  
  Some good reasons:
  - Avoid repeating code; makes program...
  - Shorter, so easier to read
  - Easier to debug (fewer occurrences of same mistake).
  - Logical organization.
Procedural Abstraction

• You write procedures all the time.

• Other than “The lab handout said I should,” why do you define a new procedure?

  Some good reasons:
  – Avoid repeating code; makes program…
    • Shorter, so easier to read.
    • Easier to debug (fewer occurrences of same mistake).
  – Logical organization.
Role of Procedures

• Procedures avoid repetition.
  Consider this sequence:
  ```java
  if (a < 0) {
      throw new IllegalArgumentException("a is negative");
  }
  if (b < 0) {
      throw new IllegalArgumentException("b is negative");
  }
  if (c < 0) // etc
  
  How can this be improved?
  ```

• Procedures are a form of abstraction by parameterization.
Role of Procedures

• Procedures avoid repetition.
  Instead of:
  ```java
  if (a < 0) {
    throw new IllegalArgumentException("a is negative");
  }
  if (b < 0) {
    throw new IllegalArgumentException("b is negative");
  }
  if (c < 0) // etc
  ```
  Can write:
  ```java
  checkNonnegative(a, "a");
  checkNonnegative(b, "b");
  checkNonnegative(c, "c"); // etc...
  ```

• Procedures are a form of abstraction by parameterization.
Role of Procedures

• Replace concrete details with abstract intent.
  Instead of:
    // Calculate area of this triangle using Heron's formula.
    double s = (a + b + c)/2.0;
    double area = Math.sqrt(s*(s-a)*(s-b)*(s-c));
  Can write:
    double area = triangleArea(a,b,c);

• But there is a catch:
  To write this, you have to know what triangleArea does.
  Procedure provides this abstraction only if it has a specification!
Role of Procedures

• Replace concrete details with abstract intent.
  Instead of:
    // Calculate area of this triangle using Heron’s formula.
    double s = (a + b + c)/2.0;
    double area = Math.sqrt(s*(s-a)*(s-b)*(s-c));

  Can write:
    double area = triangleArea(a,b,c);

• But there is a catch:
  To write this, you have to know what triangleArea does.
• Procedure provides this abstraction only if it has a specification!
Specifications (Liskov Style)

public static double triangleArea(double a, double b, double c) {
    // REQUIRES: a, b and c are all positive.
    // EFFECTS: Returns the area of a triangle with
    // side lengths a, b and c.
    // Uses Heron’s Formula.
    double s = (a+b+c)/2.0;
    return Math.sqrt(s*(s-a)*(s-b)*(s-c));
}

REQUIRES clause states conditions under which the procedure will work.

Specification is comment with “standard” format describing what
procedure does.

EFFECTS clause states what will happen (assuming REQUIRES clause is
satisfied).
public static double triangleArea(double a, double b, double c) {
    // REQUIRES: \(a, \ b\ \text{and} \ c\ \text{are all positive}\).
    // EFFECTS: Returns the area of a triangle with
    // side lengths \(a, \ b\ \text{and} \ c\).
    // Uses Heron’s Formula.

    double s = (a+b+c)/2.0;
    return Math.sqrt(s*(s-a)*(s-b)*(s-c));
}

REQUIRES clause states conditions under which the procedure will work.
“Don’t call this procedure unless…”

EFFECTS clause states what will happen (assuming REQUIRES clause is satisfied).
public static double triangleArea(double a, double b, double c) {
    // REQUIRES: a, b and c are all positive.
    // EFFECTS: Returns the area of a triangle with
    // side lengths a, b and c.
    // Uses Heron’s Formula.

    double s = (a+b+c)/2.0;
    return Math.sqrt(s*(s-a)*(s-b)*(s-c));
}

REQUIRES clause states conditions under which the procedure will work. “Don’t call this procedure unless…”

EFFECTS clause states what will happen (assuming REQUIRES clause is satisfied).
Specifications as Contracts

• Agreement between implementor and callers:
  – If you meet these conditions (REQUIRES),
    then I will do this for you (EFFECTS). (But if you don’t, forget it.)
  – Both sides have obligations.

• Choosing good specification involves tradeoffs.
  – Who benefits from more requirements? Implementor
  – Who benefits from more detailed EFFECTS clause? Caller

• When something goes wrong, contract tells you whose
  fault it is.
  – Helps you fix this problem without creating others.
Specifications as Contracts

• Agreement between implementor and callers:
  – If you meet these conditions (REQUIRES), then I will do this for you (EFFECTS). (But if you don’t, forget it.)
  – Both sides have obligations.

• Choosing good specification involves tradeoffs.
  – Who benefits from more requirements? Implementor.
  – Who benefits from more detailed EFFECTS clause? Caller.

• When something goes wrong, contract tells you whose fault it is.
  – Helps you fix this problem without creating others.
Specifications as Contracts

• Agreement between implementor and callers:
  – If you meet these conditions (REQUIRES), then I will do this for you (EFFECTS). (But if you don’t, forget it.)
  – Both sides have obligations.

• Choosing good specification involves tradeoffs.
  – Who benefits from more requirements? Implementor.
  – Who benefits from more detailed EFFECTS clause? Caller.

• When something goes wrong, contract tells you whose fault it is.
  – Helps you fix this problem without creating others.
Specifications as Contracts

• Agreement between implementor and callers:
  – If you meet these conditions (REQUIRES),
    then I will do this for you (EFFECTS). (But if you don’t, forget it.)
  – Both sides have obligations.

• Choosing good specification involves tradeoffs.
  – Who benefits from more requirements? Implementor.
  – Who benefits from more detailed EFFECTS clause? Caller.

• When something goes wrong, contract tells you whose
  fault it is.
  – Helps you fix this problem without creating others.
Specifications as Contracts

• Agreement between implementor and callers:
  – If you meet these conditions (REQUIRES),
    then I will do this for you (EFFECTS). (But if you don’t, forget it.)
  – Both sides have obligations.

• Choosing good specification involves tradeoffs.
  – Who benefits from more requirements? Implementor.
  – Who benefits from more detailed EFFECTS clause? Caller.

• When something goes wrong, contract tells you whose fault it is.
  – Helps you fix this problem without creating others.
Specifications in Debugging

Example:
Let $g$ be some procedure and suppose a program contains the line:
$$y = a[g(x)];$$
When we run program, suppose $x$ is 3 and $g(3)$ returns -1. Program **crashes** with ArrayIndexOutOfBoundsException. So where’s the error?

Three possibilities, depending on $g$’s specification:
- 3 not a valid argument for $g$.
- 3 valid, but -1 is the wrong answer for $g(3)$.
- 3 valid, -1 is the right answer.

In which case(s) is $g$ to blame for the crash?
Specifications in Debugging

Example:
Let g be some procedure and suppose a program contains the line:
\[ y = a[g(x)]; \]
When we run program, suppose x is 3 and g(3) returns -1.
Program crashes with ArrayIndexOutOfBoundsException.
So where’s the error?

Three possibilities, depending on g’s specification:
- 3 not a valid argument for g.
- 3 valid, but -1 is the wrong answer for g(3).
- 3 valid, -1 is the right answer.

*In which case(s) is g to blame for the crash?*
Specifications in Debugging

Example:
Let g be some procedure and suppose a program contains the line:
\[ y = a[g(x)]; \]
When we run program, suppose x is 3 and g(3) returns -1. Program crashes with ArrayIndexOutOfBoundsException. So where’s the error?

Three possibilities, depending on g’s specification:
- 3 not a valid argument for g.
- 3 valid, but -1 is the wrong answer for g(3).
- 3 valid, -1 is the right answer.

*In which case(s) is g to blame for the crash?* Only the second case.
There is no “right” or “wrong” without a specification.
There is no “right” or “wrong” without a specification.
A Specification Tool

• Heard of Javadoc? Used it?

• Tool for generating HTML documentation for Java classes
  – Think of the online Java API documentation
• Works by reading specially-formatted comments in code
• Most Java libraries you find online have Javadoc-generated documentation

• Sadly, Javadoc itself is not too well documented.
  – May be why Liskov doesn’t advocate using it?
• But Eclipse helps us use it fairly easily.
• Might as well use the Javadoc style for our specifications – and get nicely formatted docs for free!
A Specification Tool

• Heard of Javadoc? Used it?

• Tool for generating HTML documentation for Java classes.  
  – Think of the online Java API documentation.
• Works by reading specially-formatted comments in code.
• Most Java libraries you find online have Javadoc-generated documentation.

• Sadly, Javadoc itself is not too well documented.  
  – May be why Liskov doesn’t advocate using it?
• But Eclipse helps us use it fairly easily.
• Might as well use the Javadoc style for our specifications – and get nicely formatted docs for free!
Specifications (Javadoc Style)

/** Computes the area of a triangle using Heron’s Formula.
 * <p><b>Requires:</b> a, b and c are all positive.
 * @arg a The length of one side of a triangle.
 * @arg b The length of another side of the triangle.
 * @arg c The length of the third side of the triangle.
 * @return The area of a triangle with sides of lengths a, b and c.
 */

public static double triangleArea(double a, double b, double c) {
    double s = (a+b+c)/2.0;
    return Math.sqrt(s*(s-a)*(s-b)*(s-c));
}
Eclipse/Javadoc Demo
Tests for Procedures