Specifications

• “Without a specification your program cannot be incorrect. It can only do something unexpected. Look! A new feature!” -Gary McGraw
Software Specifications

• A specification defines the behavior of an abstraction
• This is the contract between user and provider
  – Provider’s code must implement the specification
  – Providers are free to change the implementation
    • So long as the new code still meets the specification
  – Users that depend implementation could be in trouble
    • Only rely on specification
• Black box testing essentially checks compliance of an implementation with its specification

What Makes a Good Specification?

• Sufficiently restrictive
  – Forbids unacceptable implementations
• Sufficiently general
  – Allows all acceptable implementations
• Clear
  – Easy to understand
  – A little redundancy may help (some people disagree)
Good Specifications are Hard and Rare

- Very difficult to get people to write specifications
  - Even harder to keep them up to date
- Having specifications in a separate document from code almost guarantees failure
  - Rationale for Javadoc: extract a standalone specification from the code and embedded comments
- Hard to accurately and formally capture all properties of interest
  - Always finding important details not specified

Specifications Help You Write Code

- Lots of subtle algorithms and data structures
  - Internal specs/invariants vital to correct implementation

- Example: Binary Search Tree
  - All nodes reachable from left child have smaller key than current node
  - All nodes reachable from right child have larger key than current node
Specifications Help You Maintain Code

- In the real world, much coding effort goes into modifying previously written code
  - Often originally written by somebody else
  - Perhaps six different people have modified this code
- Documenting and respecting key internal specifications are the way to avoid a mess
- Documenting and respecting key external specifications are the way to avoid having your customers storm the office with torches and pitchforks

Formal vs. Informal Specifications

```
static int find(int[] d, int x)
```

- An informal specification
  - If the array `d` is sorted, and some element of the array `d` is equal to `x`, then `find()` returns the index of `x` ……

- A formal specification
  - (for all `i`, `0 < i < d.length`, `d[i-1] < d[i]
    and there exists `j`, `0 <= j < d.length`, such that `d[j] == x`
  implies `find(d,x) = j` ……
Advantages and Disadvantages

• Formal specifications
  – Forces you to be very clear
  – Automated tools can check some specifications
    • Either at compile-time (static checking) or run-time (dynamic checking)
• Informal specifications
  – Some important properties are hard to express formally
    • Sometimes just difficult
    • Sometimes don’t have the necessary formal notation
  – Some people are intimidated by formal specs

Types of External Specifications

• Specifications on methods
  – Pre-conditions/requires: What must be true before call
  – Post-conditions/effects: What is must be true after call
    • Often relates final values to initial values

    // precondition: the array d is sorted
    // postcondition:
    //   returnValue >= 0 && d[returnValue] == x
    //   or (returnValue == -1 && x does not occur in d)
    static int find(int d[], int x);
Types of Internal Specifications

- Specifications appearing within code itself
  - i.e., comments
- Loop invariants: condition that must hold at the beginning of each iteration of a loop
  - $d[0..i]$ is sorted
- Data structure or field invariants
  - $\text{elementCount} \leq \text{elementData.length}$

Behavior vs. Function

- Side effects
  - Writes output to a file
  - Could block on a condition or mutex
- Performance
  - Should you specify performance of operations?
  - As hard as 451: what kind of bound (upper bound, amortized bound, expected bound, …), order of bound, …
  - But need at least informal specs
    - Random access is fast, insertion/deletion may be slow
Specifications and Subtyping

• Liskov substitution principle (original? formal stmt)
  – If for each object \( o1 \) of type \( S \) there is an object \( o2 \) of type \( T \) such that for all programs \( P \) defined in terms of \( T \), the behavior of \( P \) is unchanged when \( o1 \) is substituted for \( o2 \) then \( S \) is a subtype of \( T \).
  – I.e, if anyone expecting a \( T \) can be given an \( S \), then \( S \) is a subtype of \( T \).

• If we override a method, how do the specifications of the original and new method relate?

Specifications and Subtyping (cont’d)

// precondition: the array d is sorted
// postcondition:
//    returnValue >= 0 && d[returnValue] == x
//    or (returnValue == -1 && x does not occur in d)
static int find(int d[], int x);

• If we override this method, can the new method
  – Have true as a precondition?
  – Have precond “\( d \) is sorted and exists \( i \) s.t. \( d[i] == x \)”?
  – Have postcond “(returnValue==.-1 && x does not occur in d) or returnValue is first index such that \( d[returnValue] == x \)”?
  – Throw NoSuchElementException rather than returning -1 when \( x \) does not occur in \( d \)?
Javadoc

- Integrates documentation into source code as comments
- Will generate an external specification

```java
/** Javadoc Comment for this class */
public class MyClass {
/** Javadoc Comment for field text */
  String text;
/** Javadoc Comment for method setText */
  @param t Javadoc comment for parameter t */
  public void setText(String t) {...}
}
```

Javadoc example

```java
/** Given a sorted array, returns the index into the array of the given element, otherwise returns -1.

  @param d array to search in, assumed sorted
  @param x the element to search for
  @returns i >= 0 when d[i] == x, and -1 when x does not occur in d */
public static int find(int d[], int x) {
  ...
}
```
Javadoc example: HTML

Method Detail

find

public static int find(int[] d, int x)

Given a sorted array, returns the index into the array of the
given element, otherwise returns -1.

Parameters:
  d - array to search in, assumed sorted
  x - the element to search for

A Few Javadoc Tags

• Special tags for classes
  – @author
  – @version

• Special tags for methods
  – @param
  – @return
  – @exception

• Reference to another element
  – @see