Specifications & Exceptions I

Spring 2013
Today’s Topics

Specifications

- Why?
- How to decide what does what
- Java conventions for specifications

Exceptions

- Why?
- Checked vs. Unchecked
- Horrible Exception Abuse
Why Specifications?

Many bugs due to misunderstanding *interfaces between pieces of code*

**Mars Polar Orbiter**

- 327.6 million dollars
- One usable image
Why Specifications?

Many bugs due to misunderstanding *interfaces between pieces of code*

The MCO MIB has determined that the root cause for the loss of the MCO spacecraft was the failure to use metric units in the coding of a ground software file, “Small Forces,” used in trajectory models. Specifically, thruster performance data in Imperial units instead of metric units was used in the software application code titled SM_FORCES (small forces). The output from the SM_FORCES application code as required by a MSOP Project Software Interface Specification (SIS) was to be in metric units of Newton-seconds (N-s). Instead, the data was reported in Imperial units of pound-seconds (lb-s).
Why Specifications?

Many bugs due to misunderstanding *interfaces between pieces of code*

For implementer
- Freedom to change implementation (but still follow spec!)
- Easier to determine where the error is

For consumer/client
- Use methods without reading all the code
- Describe how to ensure you get what you expect
Behavioral Equivalence

```java
static int findA (int [] a, int val) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == val) return i;
    }
    return a.length;
}

static int findB (int [] a, int val) {
    for (int i = a.length - 1; i >= 0; i--) {
        if (a[i] == val) return i;
    }
    return -1;
}

Are these two the same?
```
Behavioral Equivalence

```java
static int findA (int [] a, int val) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == val) return i;
    }
    return a.length;
}

static int findB (int [] a, int val) {
    for (int i = a.length -1 ; i >= 0; i--) {
        if (a[i] == val) return i;
    }
    return -1;
}
```

Are these two the same? Can findA be used everywhere findB is used?
Behavioral Equivalence

Specifications answer the question

requires: a is nonempty
returns: index i such that a[i] is first occurrence of val
Behavioral Equivalence

Specifications answer the question

requires: a is nonempty
returns: index i such that a[i] is first occurrence of val

requires: val occurs in a
returns: index i such that a[i] = val
Preconditions/Postconditions/Frame conditions

**Preconditions (“requires”)**

- Conditions on state that must be true for the implementation to work
- Assumptions made by implementer
- If not true, then implementation behavior is undefined
  - Infinite loops, exceptions, crashes, weird/unexpected results, etc.
Preconditions/Postconditions/Frame conditions

**Preconditions ("requires")**
- Conditions on state that must be true for the implementation to work
- Assumptions made by implementer
- If not true, then implementation behavior is undefined
  - Infinite loops, exceptions, crashes, weird/unexpected results, etc.

**Postconditions ("effects")**
- What the implementer of the method promises to do
- Often, what the method returns
Preconditions/Postconditions/Frame conditions

Frame condition ("modifies")

- Identifies which objects may be modified
- … and implicitly, all other objects will not be modified
- Often omitted, meaning nothing modified
Javadoc Specifications

Preconditions → @param
Postconditions → @return
Frame conditions → include in @param clauses
Javadoc Specifications

Preconditions \(\rightarrow\) @param
Postconditions \(\rightarrow\) @return
Frame conditions \(\rightarrow\) include in @param clauses

requires: val occurs in a
returns: index i such that a[i] = val
Javadoc Specifications

Preconditions \(\rightarrow\) @param
Postconditions \(\rightarrow\) @return
Frame conditions \(\rightarrow\) include in @param clauses

requires: val occurs in exactly once in a
returns: index i such that a[i] = val

/**
 * Find value in an array.
 *
 * @param a array to search; requires that val occurs exactly
 * once in a.
 * @param val value to search for
 * @return index i such that a[i] = val
 */
Example with Mutation

```java
static int replaceA(int [] a, int val, int newVal) {
    int howMany = 0;
    for (int i = 0; i < a.length; i++) {
        if (a[i] == val) {
            a[i] = newVal;
            howMany++;
        }
    }
    return howMany;
}
```
Example with Mutation

```java
static int replaceA(int[] a, int val, int newVal) {
    int howMany = 0;
    for (int i = 0; i < a.length; i++) {
        if (a[i] == val) {
            a[i] = newVal;
            howMany++;
        }
    }
    return howMany;
}
```

requires: a has zero or more instances of val; each instance is replaced with newVal
returns: number of replacements
Example with Mutation

/**
* Replace all instances of an integer in an array.
*
* @param a array of integers. modified to replace instances
* of val with newVal
* @param val value to replace
* @param newVal replacement value
* @return number of replacements performed
*/

static int replaceA(int [] a, int val, int newVal) {
    int howMany = 0;
    for (int i = 0; i < a.length; i++) {
        if (a[i] == val) {
            a[i] = newVal;
            howMany++;
        }
    }
    return howMany;
}
INTERLUDE: JAVA CLASS HIERARCHY
Simple way to express class hierarchy relationships

- We will use these occasionally to talk about associated classes
List<Integer> xPoints = new ArrayList<Integer>();
Java & Class Hierarchy

In Java, variables of a parent type can hold an object of a child type.

List<Integer> xPoints = new ArrayList<Integer>();

But?
EXCEPTIONS
“Special” Values

```java
static int findB (int [] a, int val) {
    for (int i = a.length -1 ; i >= 0; i--) {
        if (a[i] == val) return i;
    }
    return -1;
}
```

Why return -1?

- What if there’s no “special value”
- E.g. Date

Can use an exception instead
“Special” Values

```java
static int findB (int [] a, int val) throws NotFoundException {
    for (int i = a.length - 1 ; i >= 0; i--) {
        if (a[i] == val) return i;
    }
    throw new NotFoundException();
}
```

**Why return -1?**

- What if there’s no “special value”
- E.g. Date

**Can use an exception instead**
Java Exceptions

Two kinds of exceptions: Checked and Unchecked
Java Exceptions

Two kinds of exceptions: Checked and Unchecked

Checked

➢ Used for special results

➢ Compiler requires method signature state that may throw checked exception

➢ Caller must either also declare it throws that exception, or handle it
Java Exceptions

Two kinds of exceptions: Checked and Unchecked

Checked

➢ Used for special results
➢ Compiler requires method signature state that may throw checked exception
➢ Caller must either also declare it throws that exception, or handle it

```java
int myMethod() throws SomeException {...}

void foo() {
    try {
        myMethod();
    } catch (SomeException e) {...}
}

void foo2() throws SomeException {
    myMethod();
}
```
Java Exceptions

Two kinds of exceptions: Checked and Unchecked

Unchecked

- Used for unexpected/catastrophic failures
- E.g. NullPointerException: “Thrown when an application attempts to use null in a case where an object is required.” [Java Platform Docs]
- Compiler does not check that methods declare these
- Compiler does not check that callers declare or catch unchecked exceptions
What’s Unchecked vs Checked?

 Depends on where in the class hierarchy of Throwable.

```
Throwable
  ↓
Exception Error
  ↓
RunTimeException
```
What’s Unchecked vs Checked?

Depends on where in the class hierarchy of Throwable.

Throwable

Exception

Error

RunTimeException

Reserved for internal JVM errors*
What’s Unchecked vs Checked?

Depends on where in the class hierarchy of Throwable.

- Throwable
- Exception
- RunTimeException
- Error
- Reserved for internal JVM errors*

Unchecked exceptions
What’s Unchecked vs Checked?

 Depends on where in the class hierarchy of Throwable.

© 6.005 Course Staff 07-13
What’s Unchecked vs Checked?

Depends on where in the class hierarchy of Throwable.

- **Checked** exceptions:
  - Exception
    - RunTimeException
  - Error
    - Reserved for internal JVM errors*

- **Unchecked** exceptions:

Always subclass Exception or RunTimeException (or one of their children) when creating your own exception types!

*Reserved for internal JVM errors*
Exception Abuse

```java
try {
    int i = 0;
    while (true) a[i++].f();
} catch (ArrayIndexOutOfBoundsException e) { }
```

What is this code doing?
Exception Abuse

```java
try {
    int i = 0;
    while (true) a[i++].f();
} catch (ArrayIndexOutOfBoundsException e) { }
```

What is this code doing?

```java
for (int i=0; i<a.length; i++)
a[i].f();
```
Exception Abuse

```java
try {
    int i = 0;
    while (true) a[i++].f();
} catch(ArrayIndexOutOfBoundsException e) { }
```

**What is this code doing?**

```java
for (int i=0; i<a.length; i++)
a[i].f();
```

**While we’re at it, why do Java libraries sometimes return null?**

➢ We’ll get into this next time
Why Not Specifications?

4. My code is self-documenting (aka the specs are OBVIOUS!)
Why Not Specifications?

4. My code is self-documenting (aka the specs are OBVIOUS!)
3. I’ll do it later.
Why Not Specifications?

4. My code is self-documenting (aka the specs are OBVIOUS!)
3. I’ll do it later.
2. It’s going to change ANYWAY.
Why Not Specifications?

4. My code is self-documenting (aka the specs are OBVIOUS!)
3. I’ll do it later.
2. It’s going to change ANYWAY.
1. The specification is in my head (DUH!)
Why Not Specifications?

4. My code is self-documenting (aka the specs are OBVIOUS!)
3. I’ll do it later.
2. It’s going to change ANYWAY.
1. The specification is in my head (DUH!)
0. We’re AGILE!