IMPORTANT NOTE

Some parts of these section slides deal with null ints. This was a mistake, as primitives cannot be null. These issues have been corrected.

CSE 331 AUT18

Section 1: Intro and Specifications

IntelliJ Setup

- Homework will be posted later today
- Instructions for setup are posted
- If you try to follow them but run into problems, please come to office hours!

Welcome to section!

- We meet once a week on Thursdays
- Different TAs teach section each week
- Section is not optional!
  - Section is a supplement to lecture
  - It gives you a chance to practice the material and make sure you understand it
  - Sometimes, section may contain material that is not in lectures
What is a specification?

• How you tell the client what your code does

• A "contract" between the developer and the user
  • The developer promises to fulfill the specification
  • The user agrees to only rely on functionality defined in the specification

Why do we need Specifications?

• Other people use your code!
  • Is it a good idea to share all of your source code with everyone who uses it?

• You don’t have a perfect memory
  • Can you remember the details of a program you wrote 6 months ago?

• They encourage easy and understandable code

Format of a Specification

/**
* @spec.requires
* @spec.modifies
* @spec.effects
* @return
* @throws
**/
methodName {...}

Format of a Specification

/**
* @spec.requires
* @spec.modifies
* @spec.effects
* @return
* @throws
**/
methodName {...}

PRECONDITION
• What your method requires to be true before it is called
• If the precondition is not met, there are no guarantees on the method's behavior

POSTCONDITION
• Guarantees the implementor makes about the program state after the method is called
• If the preconditions are satisfied, the postconditions must hold
Format of a Specification

/**
 * @spec.requires
 * @spec.modifies
 * @spec.effects
 * @return
 * @throws
 */

methodName {…}

PRECONDITION
• What your method requires to be true before it is called
• If the precondition is not met, there are no guarantees on the method's behavior

POSTCONDITION
• Guarantees the implementor makes about the program state after the method is called
• If the preconditions are satisfied, the postconditions must hold

Satisfying Specifications

• An implementation M satisfies a specification S if:

1. When all the preconditions of S are met,
2. All the postconditions of S are satisfied by M after it executes
Satisfying Specifications

```java
/** Computes the area of a rectangle with width w and length l
 * @returns the area of a rectangle with width w and length l
 */

public int area(int w, int l) {
    if (w < 0 || l < 0) {
        throw new IllegalArgumentException();
    }
    return l * w;
}
```

Does the implementation satisfy this specification?

No! – The specification is violated when w and/or l are negative

Satisfying Specifications

```java
/** Computes the area of a rectangle with width w and length l
 * @returns the area of a rectangle with width w and length l
 * @throws IllegalArgumentException if w < 0 or l < 0
 */

public int area(int w, int l) {
    if (w < 0 || l < 0) {
        throw new IllegalArgumentException();
    }
    return l * w;
}
```

Does the implementation satisfy this specification?

Yes! – The specification matches the implementation
Satisfying Specifications

```java
public int area(int w, int l) {
    if (w < 0 || l < 0) {
        throw new IllegalArgumentException();
    }
    return l * w;
}
```

Does the implementation satisfy this specification?

Yes! – Even though we didn’t document the `IllegalArgumentException` in the `@throws` tag, since we require `w` and `l` to be non-negative, we aren’t obligated to specify what happens when the user enters a negative `w` or `l`.

Stronger vs Weaker Specifications

- A specification R is stronger than another specification S if:
  - Every implementation that satisfies R also satisfies S
  - R has a weaker precondition and/or a stronger postcondition

Both of these definitions are equivalent!
Stronger vs Weaker Preconditions

- A precondition is weaker when it requires less from the user:
  - Less requirements in the `@spec.requires` tag
- Which specification has a weaker precondition?

```plaintext
/**
 * @spec.requires x > 0
 * @return x
 */
```

```
/**
 * @return x if x > 0, -x if x <= 0
 */
```

Stronger vs Weaker Postconditions

- A postcondition is stronger when it makes more guarantees on the final program state after execution

```plaintext
/**
 * @spec.requires x > 0
 * @return x
 */
```

```
/**
 * @return x if x > 0, -x if x <= 0
 */
```

```plaintext
/*
 * @return x if x > 0, -x if x <= 0
 */
```

```
/*
 * @return x if x > 0, -x if x <= 0
 */
```

Stronger vs Weaker Postconditions

- A postcondition is stronger when it makes more guarantees on the final program state after execution
  - More objects in the `@spec.modifies` tag
  - `@spec.effects` is harder to satisfy
  - `@returns` is harder to satisfy
  - Use a subtype of an exception in `@throws`
Stronger vs Weaker Postconditions

• A postcondition is stronger when it makes more guarantees on the final program state after execution
  • Less objects in the @spec.modifies tag
  • @spec.effects is harder to satisfy
  • @returns is harder to satisfy
  • @throws

• Which specification has the stronger postcondition?

```java
/**
 * @spec.requires x > 0
 * @returns x
 **/
```

```java
/**
 * @spec.requires x > 0
 * @returns x if x > 0, -x if x <= 0
 **/
```

Which specification has the stronger postcondition?

```java
/**
 * @spec.requires x > 0
 * @return x
 **/
```

```java
/**
 * @spec.requires x > 0
 * @return x if x > 0, -x if x <= 0
 **/
```

Both have different behavior for x <= 0. They are both incomparable.

---

Stronger vs Weaker Specifications

Which specification is stronger?

```java
/**
 * @return x
 * @throws IllegalArgumentException if x <= 0
 **/
```

```java
/**
 * @return x if x > 0, -x if x <= 0
 **/
```

Which specification is stronger?

```java
/**
 * @return s
 * @throws IllegalArgumentException if s == 0
 **/
```

```java
/**
 * @return s if s > 0, -s if s <= 0
 **/
```

Both have different behavior for s <= 0. They are both incomparable.
Worksheet