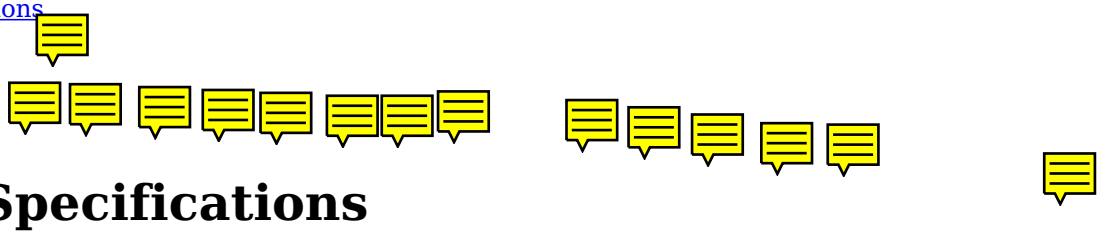

Reading 6: Specifications

Software in 6.005

Safe from bugs	Easy to understand	Ready for change
Correct today and correct in the unknown future.	Communicating clearly with future programmers, including future you.	Designed to accommodate change without rewriting.

Objectives

- Understand preconditions and postconditions in method specifications, and be able to write correct specifications
- Be able to write tests against a specification
- Know the difference between checked and unchecked exceptions in Java
- Understand how to use exceptions for special results

Introduction

Specifications are the linchpin of teamwork. It's impossible to delegate responsibility for implementing a method without a specification. The specification acts as a contract: the implementer is responsible for meeting the contract, and a client that uses the method can rely on the contract. In fact, we'll see that like real legal contracts, specifications place demands on both parties: when the specification has a precondition, the client has responsibilities too.

In this reading we'll look at the role played by specifications of methods. We'll discuss what preconditions and postconditions are, and what they mean for the implementor and the client of a method. We'll also talk about how to use exceptions, an important language feature found in Java, Python, and many other modern languages, which allows us to make a method's interface safer from bugs and easier to understand.

Part 1: Specifications

Part 2: Exceptions

Summary

Before we wrap up, check your understanding with one last example:

A specification acts as a crucial firewall between the implementor of a procedure and its client. It makes separate development possible: the client is free to write code that uses the procedure without seeing its source code, and the implementor is free to write the code that implements the procedure without knowing how it will be used.

Let's review how specifications help with the main goals of this course:

- **Safe from bugs** . A good specification clearly documents the mutual assumptions that a client and implementor are relying on. Bugs often come from disagreements at the interfaces, and the presence of a specification reduces that. Using machine-checked language features in your spec, like static typing and exceptions rather than just a human-readable comment, can reduce bugs still more.

- **Easy to understand** . A short, simple spec is easier to understand than the implementation itself, and saves other people from having to read the code.

- **Ready for change** . Specs establish contracts between different parts of your code, allowing those parts to change independently as long as they continue to satisfy the requirements of the contract.