Recitation 9

Brandon Rozek
rozekb@rpi.edu

Rensselaer Polytechnic Institute, Troy, NY, USA

March 2022

Outline

Two Things:

- Polymorphism
- SOLID

Overriding vs Overloading

Recall from Recitation 3...

- *Overloaded methods* is a language feature that allows for the same method name with different argument types. These are bound with statically.
- *Overridden methods* changes the method called when a subclass uses the same method name as the superclass. These methods are bound dynamically. Argument types must be the same but return type may differ.

Question:

1. What is static binding?
2. What is dynamic binding?

Method Call Execution

- What happens under the hood when you call a method?
- We'll need to consider what happens in *compile-time* and *run-time.*

Compile Time Method Determination

Consider the following example: `p1.playCard(5);`

1. Determine the method's class: `Player`
2. Determine the method signature: `void playCard(int)`
3. Find all methods in the class that matches the signature.
   - This includes inherited methods
   - Input arguments can be subtypes of found signatures
4. Discard methods that are not accessible
5. Keep most specific signature

Run Time Method Determination

1. Determine the runtime type of `p1` by checking the heap
2. Until the method is found:
   1. Find the method that matches the method call's signature.
   2. If found, invoke.
   3. Otherwise, check the superclass.

Note: This is guaranteed to find a method since it passed compilation.

Polymorphism

- *Polymorphism* provides one interface to data of a different type.
- Ad-Hoc Polymorphism
  - Functions that behave differently depending on the arguments supplied.
  - Ex: Overloading
- Parametric Polymorphism
  - Types that are not specified but can represent any type.
  - Ex: Java Generics, C++ Templates
- Subtype Polymorphism
  - Where a subtype relates to a supertype by some notion of substitutability.
  - Ex: Overrides

Question:

What type of polymorphism are the following featuring?

1. `ArrayList<int> x = new ArrayList<int>();`
2. `p1.equals(p2);`
3. `Brandon.say("Hi"); Brandon.say("Hi␣Bob", bob);`

SOLID

Single Responsibility Principle: Every class should have one job.

Open-Closed Principle: Software entities should be open for extension but closed for modification.

Liskov Substitution Principle: An object with stronger specification can be substituted for an object with a weaker one without altering correctness.

Interface Segregation Principle: Many client-specific interfaces are better than one general-purpose interface.

Dependency Inversion Principle: Depend upon abstractions, not concretions.

Subclassing

Inheriting a class in Java is the same as subclassing.

Benefits:

- Reuse of code: fields and methods
- Simpler Maintenance: Fix issues only in superclass

Disadvantages:

- May break one of the equivalence relation properties.
- Fragile Base Class Problem: Changes in the implementation details of the superclass can break subclasses.

True Subtypes

A class B is a true subtype if:

- B is a subclass of A AND
- B has a stronger specification than A

Question:

1. What makes one specification stronger than another?
2. Are Java subtypes true subtypes?

Liskov Substitution Principle

One good software design practice is to ensure that every subclass is a true subtype of its superclass.

How? Given a subclass B of A:

- B should not remove methods from A
- For each method $B.m$ that override's $A.m$, $B.m$ must not have a weaker spec than $A.m$.
- Any property guaranteed by supertype must be guaranteed by subtype

Contravariance and Covariance Review

Review from Recitation 5...
Lets say we have classes `Student` and `Person` where `Student` is a subtype of ($<:$) `Person`.

Now consider the composite classes `C<Student>` and `C<Person>`:

- The relationship is *covariant* if `C<Student>` $<:$ `C<Person>`
- The relationship is *contravariant* if `C<Person>` $<:$ `C<Student>`
- *Bivariant* is both covariant and contravariant.
- *Invariant* is neither covariant nor contravariant.

Question:

What are the variances of the following?

1. Java Arrays?
2. Java Generics?

Liskov Substition Principle Rules

- Parameter types of A.m may be replaced by supertypes in subclass B.m.
- Return type of A.m may be replaced by subtype in subclass B.m
- No new exceptions should be thrown, unless the exceptions are subtypes of exceptions thrown by the parent
- Preconditions cannot be strengthened in the subtype.
  - You cannot require more than the parent
- Postconditions cannot be weakened in the subtype.
  - You cannot guarantee less than the parent

Substition Principle for Classes

Constraints on classes:

- Any property guaranteed by supertype must be guaranteed by subtype
- Subtype can strengthen and add properties
- Anything provable about A is provable about B
- A's rep invariant must hold in B
- No specification weakening
- No method removal

Substitution Principle for Methods

Constraints on methods:

- May introduce new methods
- Each override method must have a stronger or equal spec
- Weaker or equal precondition / Ask nothing extra of client
- Guarantee as much as supertype
- Effects clause is at least as strict as supertype
- No new entries in modifies clause
- The overriding method satisfies the supertype spec
- No new exceptions in domain

Question:

1. If class A throws `Error`, can a true subtype B throw `AssertionError`?
2. If class A guarantees $balance \geq 0$, can a true subtype B guarantee $balance \geq -1$?
3. Given a class A, can a true subtype B have less methods?

Any Questions?