Activity 7-2: The Liskov Substitution Principle

Types

What are types, really?

Video 1: 20:00-21:56

- A type is a "bag of operations".
- Classes (and interfaces) are basically types: The methods specify what operations can be performed on objects of that "type".

Video 1: 21:56-24:40

- Data structures are not operations.
- Relation between classes and subclasses: point and described point.
- A **subtype** can be used in place of its parent type.

Liskov Substitution Principle

Video 1: 25:50-28:00

- "Subtypes should be substitutable for their parent types".

Video 1: 29:00-32:10

- Duck typing: In dynamically typed languages, can still have subtypes of other types, via duck typing.

Refused Bequests

Video 1: 32:10-32:30 Video 1: 33:06-34:24

- When LSP is violated, we usually have a "refused bequest": We expect the object to respond to a method, but the object doesn't have that method (usually throws an exception).
- More generally, a refused bequest happens when the object of a subclass does something that the users of the superclass did not expect.

Video 1: 34:24-38:36

- The Rectangle and Square classes.

Video 1: 38:36-40:40

- The problem with the expectations of the users of Rectangle.

Video 1: 40:40-42:00

- LSP violations are latent violations of the OCP

Video 1: 42:00-43:50

- Some solutions that don't quite work out.

Video 1: 43:50-46:14

- Solution: don't expect a subtype relationship between rectangles and squares
- The Representative Rule: The representatives of objects don't share the relationships of the objects they represent.

Video 1: 48:20-50:10

- The problem with subtypes and lists.

Heuristics for knowing if you violate LSP

Video 2: 00:30-06:10

- Subtypes can do more, but cannot do less
- Be suspicious of degenerate implementations
- Derived functions that throw exceptions are almost always violations of LSP
- "if instanceof" statements are indications of violation of LSP
- you are allowed to check the type of an instance, if you already know what type it is (but the compiler has forgotten it)
- typecases should be replaced by polymorphic dispatch

Static vs Dynamic languages

Video 2: 06:10-10:20

- Inheritance gives both flexibility and rigidity
- TDD can help detect the type errors that static typing would detect

An example: The modem problem

Video 2: 13:48-17:40

- The Modem interface and the FileMover programs
- Dedicated modem class
- Making ded modem extend modem

Video 2: 17:40-23:55

- Problems with the extension

Video 2: 23:55-25:20

- Long-distance fragility as a consequence of the LSP violation

Video 2: 25:20-27:30

- Solution via adapter