Designing for Inheritance & Exceptions

Ch 6
Topics

1) How can we **correctly** use inheritance?
2) How can we use **exceptions**?
Designing for Inheritance

If all you have is a hammer, everything looks like a nail.
-- Abraham Maslow, 1966
Tips for Inheritance...

- Inheritance is only for..
- Maximize reuse of design and code
  - ..
- To ensure common interface in all derived classes, give base-class abstract methods.
- Encapsulation:
  - use `super` in constructors and for overridden methods.
  - use visibility modifiers to provide sufficient access but maintain encapsulation.
  - avoid `protected`: fields should be `private`.
When to use Inheritance?

- What is sufficient grounds to use inheritance?
  - Code reuse?
  - Is-a relationship?
  - Shared public interface?
  - Polymorphism?
Reason 1: Code Reuse

• **Idea**: Inherit shared functionality from a base class.

  - Human & Dog have duplicate code (fields & methods), but..

• **Limitation**

  - (Could create a “NamedMammal” base-class)
Is-A

- **Idea:** Represents a...
- **Example:**
  - Square is-a Rectangle, and gives reuse.
  - But..

- How can we describe this problem?

What is an example method in Rectangle inconsistent with Square?
Is-A: LSP

- Liskov Substitution Principle (LSP)  
  B can inherit from A only if:

  1) that A's method accepts (or more) and
  2) that A's method does (or more).

- What methods in Rectangle fail LSP for Square?
  - **Square** does not do the same things with all values as **Rectangle**: fails LSP.
Is-A: LSP & Immutable

- LSP & Immutable
  - Would making Rectangle and Square immutable help?

- Is-A Limitation: Must..
Is-A LSP: Example

- Photographer can photograph any Animal. DuckPhotographer only wants to photograph Ducks.
- DuckPhotographer.photograph() wants to reject non-ducks
  - Could throw an IllegalArgumentException?
- DuckPhotographer
  - ..
  - ..
Is-A LSP

- **Rephrase LSP:**
  - Client code using a reference to the base class must be able to..

  - i.e., behaviour is unchanged.
LSP is part of a common set of 5 OOD principles:

- **S** SRP: Single Responsibility Principle
  “Class has one responsibility”

- **O** OCP: Open Closed Principle
  “Be open for extension, closed for modification”

- **L** LSP: Liskov Substitution Principle
  “Subtype objects interchangeable with base objects”

- **I** ISP: Interface Segregation Principle
  “Favour many client specific interfaces”

- **D** DIP: Dependency Inversion
  “Depend on abstractions, not concrete classes”
Public Interface

- **Idea:** Share an..

- **Example:**
  - What about when a student..
  - Cannot..

- **Limitation:**
  Don't use inheritance for..
  - Use reference vs inheritance.
Polymorphism

- **Idea:** Access derived classes through..

- **Inheritance is..** shared code, is-a, shared interface, polymorphism.

- **Example:** New TextBox inherit Rectangle.
  - Share code:
  - Is-a:
  - Shared Interface:
  - Polymorphism:
Avoiding Inheritance?

- Consider using..

Ex:

- Square.. private Rectangle field.
- Use the rectangle's code without copying it.
- Write the public interface you want without worrying about a base-class interface.
- But..
Decisions

- **Inheritance Design**
  - Think through inheritance hierarchy before coding.
  - Which is best?

- **Decide:**
  - Where is..
  - Where is..
  - Place it as high up (base class) as possible.
Exceptions
Exceptions

- **Try-block:** Code to check for an exception.
- **Catch-block:** Handle possible exceptions.
  - Check if thrown exception matches or is...

```java
public static void main(String[] args) {
    String input = "123xxx";
    try {
        int num = Integer.parseInt(input);
        System.out.println("That's the number "+num);
    } catch (NumberFormatException e) {
        System.out.println("Bad input.");
    }
}
```
private void foo() {
    try {
        // May throw exceptions
        callA();
    } catch (T1 exception) {
        callB();
    } catch (T2 exception) {
        callC();
    } catch (T3 exception) {
        callD();
    }
    // Some more code
    callZ();
}
Try-Catch Example

```java
void tryCatch() {
    double[] data = new double[]{};

    try {
        double avg = average(data);
        System.out.println("Average value: " + avg);
    } catch (IllegalArgumentException ex) {
        System.out.println("Unable to compute: " + ex.getMessage());
    }
}

double average(double[] data) {
    if (data.length == 0) {
        throw new IllegalArgumentException("Array must not be empty.");
    }

    double sum = 0;
    for (double val : data) {
        sum += val;
    }

    return sum / data.length;
}
```

= 14. SimpleTryCatch.tryCatch()
The **finally** Clause

- **Finally clause..**
  - Optional clause after all the `catch` clauses.

- **Execution Possibilities**
  - **No exception:**
    `finally` block executed..
  - **Exception in try is un-caught:**
    `finally` block executed..
    the statement which threw the exception.
  - **Exception in try is caught:**
    `finally` block executed..
    which catches that exception type.

- Often used for clean-up code (close file).
private void foo() {
    try {
        // May throw exceptions
        callA();
    } catch (T1 exception) {
        callB();
    } catch (T2 exception) {
        callC();
    } finally {
        callD();
    }
    // Other code to do after...
    callZ();
}

// Other code to do after...
void tryFinally() throws IOException {
    double[] data = new double[]{};

    FileWriter fw = null;
    try {
        fw = new FileWriter("someData.txt");

        double avg = average(data);
        fw.write("Average value: "+ avg);
    } finally {
        // Close the file, no matter what.
        fw.write("Encountered error... closing output file!");
        fw.close();
    }
}

double average(double[] data) {
    if (data.length == 0) {
        throw new IllegalArgumentException("List must not be empty.");
    }
    
    // .... return ....
}
Exception Propagation

- **Uncaught exception:** execution immediately..
  (after finally).
  - Propagates up until a method catches exception.
  - If `main()` does not handle it, the program is terminated.

- **Exception handling is a design decision.**
  - Could..
    - Could have one of the calling methods handle it.
    - Could even let it terminate the program.

- **Example:**
  - Allow exception in database code..
    to catch and display meaningful error.
public class HappyCode {
    public static void main(String[] args) {
        level1();
    }
    static void level1() {
        try {
            level2();
        } catch (ArithmeticException e) {
            e.printStackTrace();
        }
    }
    static void level2() {
        level3();
    }
    static void level3() {
        int a = 1 / 0;
    }
}
Re-throwing Example

```java
void tryRethrow() {
    double[] grades = new double[]{};
    double avg = getAverageGrade(grades);
    System.out.println("Average grade: " + avg);
}

private double getAverageGrade(double[] grades) {
    try {
        return average(grades);
    } catch (IllegalArgumentException ex) {
        // Wrap the exception is another exception
        throw new IllegalStateException("No grades entered", ex);
    }
}

double average(double[] data) {
    if (data.length == 0) {
        throw new IllegalArgumentException("List must not be empty.");
    }
    // .... return ....
}
```
Exception Class Hierarchy
Exception Class Hierarchy

- **All exceptions inherit from..**
  - Many high-level exceptions in `java.lang` package.
  - Custom exceptions inherit from these high-level classes.

- **Some methods in Exception**
  - `String getMessage()`
    Returns a string which describes the exception.
  - `void printStackTrace()`
    Prints the stack trace to `System.err` (error).
  - `void printStackTrace(PrintStream s)`
    Prints the stack trace to the given `PrintStream s`. 

Error & Exception Hierarchy (part)

Object
- Throwable
  - Exception
    - RuntimeException
      - ArithmeticException
      - IndexOutOfBoundsException
      - NullPointerException
      - IllegalAccessException
      - IOException
      - IllegalArgumentException
      - NumberFormatException
    - IOException
      - FileNotFoundException
    - HardwareException
      - LinkageError
      - ThreadDeath
      - VirtualMachineError
      - AWTError

Plus MANY more! Even create your own
Checked vs Unchecked Exceptions

• **Checked Exceptions**
  - Must be either.. or must be..
  - This acknowledges that an exception can be thrown.
    ```java
    int foo() throws FileNotFoundException {
        ...
    }
    ```

• **Unchecked Exceptions**
  - in throws clause.
  - `RuntimeException` or its derived classes are *unchecked*
    All other exceptions are *checked*. 
**Checked vs Unchecked**

- **Check vs unchecked exceptions**
  - Unchecked make for cleaner...

```
public class DemoCheckedExceptions {
    public void top() throws FileNotFoundException {
        foo1();
    }

    public void foo1() throws FileNotFoundException {
        foo2();
    }

    public void foo2() throws FileNotFoundException {
        throw new FileNotFoundException();
    }
}
```

```
public class DemoUncheckedException {
    public void top(String[] args) {
        foo1();
    }

    public void foo1() {
        foo2();
    }

    public void foo2() {
        throw new NullPointerException();
    }
}
```

- **Prefer unchecked exceptions**
  Can change which exceptions are thrown without..

```
public class DemoUncheckedException {
    public void top(String[] args) {
        foo1();
    }

    public void foo1() {
        foo2();
    }

    public void foo2() {
        throw new NullPointerException();
    }
}
```
Error & Exception Hierarchy (part)

Object

Throwable

Exception

RunnableException

ArithmeticException

IndexOutOfBoundsException

NullPointerException

UncheckedExceptions and Errors

LinkageError

ThreadDeath

VirtualMachineError

AWTError

Error

Everything else is checked.

IllegalAccessException

IOException

FileNotFoundException

IllegalAccessException

NullPointerException

IndexOutOfBoundsException

ArithmeticException
Custom Exceptions

- Create your own exceptions by..

- A new exception class allows code to specifically catch the errors your code sends.

```java
/**
 * Indicates that no file was selected.
 */
public class NoFileSelected extends RuntimeException {
    public NoFileSelected() {
        super();
    }
    public NoFileSelected(String message) {
        super (message);
    }
}
```
throw

- You can explicitly throw an exception object:
  ```java
  String getFile() {
    if (fileName == null) {
      throw new NoFileSelected("File not selected.");
    }
    return fileName;
  }
  ```

- As a designer, you choose:
  - throw an exception?
  - return a “failure” status (such as `false` or -1)?
  - try to correct the data (error recovery)
Clean Exceptions

- Exception handling can really complicate code.
- **Suggestion:**
  - `fooThrows()`: does the work, but no exception handling.
  - `foo()`: call's `fooThrows()` then does exception handling.

**Original Code**

```java
void foo() {
    try {
        // do something complicated
        // which throws exception
        for(...) {
            if(...)
                throw new DaUhOh();
        }
    } catch (SomeException e) {
        showUser("Oops...");
    }
}
```

**Refactored Code**

```java
void foo() {
    try {
        fooThrows();
    } catch (DaUhOh e) {
        showUser("Oops...");
    }
}
```

```java
void forThrows() throws DaUhOh {
    for(...) {
        if(...)
            throw new DaUhOh();
    }
}
```
Java 7's Exceptional Enhancements
Resources

• Some resources must be freed:
  – Ex: Scanner's close() must be called to avoid a resource leak.
  – Can use.. to always close resource.
Try-With-Resources

- Java 7 (1.7) supports try-with-resources
  - Can declare scanner *inside* the try ( )
  - Significantly cleans up code!
- Works for objects implementing AutoClosable or Closable Interfaces.

```java
/**
 * Read a number from a file using try-with-resources.
 */
public static void readNum(String fileName)
    throws FileNotFoundException {
    File file = new File(fileName);
    try (Scanner scanner = new Scanner(file)) {
        if (scanner.hasNextInt()) {
            int num = scanner.nextInt();
            System.out.println("# "+ num);
        }
    }
}
```

DemoTryWithResource.java

20-03-23
Summary

- Use inheritance only when called for by all of:
  - code reuse
  - is-a relationship & LSP
  - public interface
  - polymorphism

- Exceptions
  - try-catch-finally & exception propagation.
  - Checked vs unchecked exceptions
  - exception inheritance hierarchy & own exceptions.
  - clean exception code: unchecked & tryThrows()
  - Java 7: try-with-resources