Lecture 1

Nyasha Mufudi

Outline

- OOP and Procedural Programming
- Encapsulation
- Inheritance
- Polymorphism

References

- Lafore, Robert R. Data **Structures and Algorithms**, chapter 2
- Gamma, Erich et al. Addison-Wesley **Design Patterns Elements of Reusable Object-Oriented Software**
- Furlong, John. **Program development in Java**, chapter 1-4

OOP and Procedural Programming

OOP and Procedural Programming

- Procedural programming uses a list of procedures to tell the computer what to do step by step. It has worked well for short programs, but as programs have become more complex, procedures have become impossible to manage.
- Object-oriented programs organize memory with each other to achieve the program's intended purpose.

OOP and Procedural Programming

- Requirement: Print out the shapes on left, a square, a circle, and a triangle, in a program that allows the user to specify which shape to create and to draw the shape. The program should also be able to erase the shape when the user is finished with it.

OOP and Procedural Programming

- There is a requirement longer. Draw and Erase procedures should know the type and the shape. Write code that will do this.

- Problem: Code is longer and harder to write.
- Solution: Use a procedure to determine the shape.
- Procedure Draw() and Erase() are longer and harder to write.
- Procedure Draw() and Erase() are harder to understand.
- Procedure Draw() and Erase() are harder to debug.
- Procedure Draw() and Erase() are harder to modify.

OOP and Procedural Programming

OOP and Procedural Programming

- Draw
- Erase
- OOP can more easily do this.
- OOP can operate on data more easily.
- OOP can more easily model real-world objects.
- OOP increases level of code by isolating interfaces and implementations.

Encapsulation

Get, Set method and Encapsulation

- Imagine you want the code to a class and another dever wanted to know the current values of some of your attributes.

Encapsulation

Encapsulation

- Advantage: implementation changes.
- Requires a change to client code.

Encapsulation

- Implementation variables hidden (with an access modifier, often private).
- Make public accessor methods, and fine-tuning code to use those. Client code will continue to work unchanged, even if you change the implementation of the class, or even if you change the set of attributes in the class.
- For those access methods, use the best conventions naming conventions and compiler optimizations.

Inheritance

Inheritance

- Imagine you need to code an app to control an organization. You need to keep track of employees, managers, and CEOs. Managers and CEOs have all the properties of employees, plus some extra ones.

Inheritance

- Those attributes may have different properties. A programmer may be tempted to cut and paste the code for Employee, and create a Manager class by cut-and-pasting and adding new attributes.

Problem with the design

Inheritance

- All of the common properties only in one position (a new class, called parent class) and inherit them.

Inheritance

Inheritance

- Adding new attributes is easy.
- Inheritance is a powerful way to eliminate **redundancy**.
- Inheritance is also called **"is-a-type-of"** or **"is-a-kind-of"**. Every **Manager** is an **Employee**, and every **CEO** is an **Employee**. Every **Employee** is a **CEO**. **Subtypes** are related classes that can **inherit** from a **supertype**. Supertypes are also called **base classes**.
- Subtypes inherit within a hierarchical parent, known as an inheritance tree or type hierarchy.

Type hierarchy - Lion

Type hierarchy - Exceptions

IS-A and has-A Relationships

- IS-A: The concept of a Manager is a kind of Employee.
- Implementation: Manager class inherits from the Employee class.
- Has-A: HAS-A relationships are based on a part, rather than inheritance.

Polymorphism

Polymorphism

- The word **"polymorphism"** means "many forms".
- Polymorphism is the ability **a set object to exhibit different behaviors** depending on the context.
- Attack the **Superclass pointer contract** to **Jedi**?

Reference variable

- The only way to access an object is through a reference variable.
- Reference variables can be declared as classes that are compatible with the type of the reference variable. A reference variable can be declared as a class or as a subclass of that class.
- An reference is portable, and can be reassigned to other objects that are not of the same type as the original object.
- Reference variables are strongly typed. You can only assign to a reference variable a compatible object.
- An reference variable can refer to any object of the same type as the declared type, or to a subtype of the declared type. A reference variable can also refer to any object that **implements** the declared interface.
- **Reference variables can refer to any object that implements the declared interface. Reference variables can refer to any object that has the implements the interface.**

Substitution Principle

- An object of a supertype should be replaceable with objects of its subtypes without affecting the correctness of the program.

Substitution Principle

- The substitution principle requires that the subtype specification **strengthen** the supertype specification. Subtypes must have properties that are **stronger**.
- Subtypes may:
  - Add properties
  - Strengthen properties

Signature Rule

The subtype requires must have all the methods of the supertype, and the signatures of the methods must be compatible.

Contravariance: the argument types of the subtype must be more general than those of the supertype.

Covariance: the return type of the subtype must be more specific than that of the supertype.

- Contravariance: $T_{sub} \leq T_{super}$
- Covariance: $T_{sub} \leq T_{super}$

Properties rule

- Properties are a special kind of properties that can be stressed under certain conditions.
- Example: A supertype method forces an attribute $0 \leq x \leq 10$. A subtype method requires that $0 \leq x \leq 5$.

The following is the current exercises and is a subtype

(Percent: operation at 0.0,0.0,0.0: exercises on 0)

$$(0 \leq 0) \quad 0 \leq 10 \quad (0 \leq 10 \leq 0 \leq 10)$$

Java's dynamic method dispatching

- Java takes advantage of the fact that the right version of an object's method is called at runtime.
- It is how Java implements runtime polymorphism.
- The implementation finds the correct type (runtime type) of the variable call the choices.

Benefits of Polymorphism

- Increase code reusability by allowing classes of different classes to be treated in the same way.
- Increase code flexibility and extensibility, of parts by relying the correctness of the program on the substitution principle.
- Separate dynamic behavior, making the commonalities of the called of runtime based on the type and value of the object.
- Simplify the code by using the same method call to the subtypes to invoke polymorphic code (an example dispatch different types).

Summary

- OOP and Procedural Programming
- Encapsulation
- Inheritance
- Polymorphism