Soyeon Kim, 106292803

Homework 8

CS 131, Fall 2024
Carey Nachenberg

OOP1: Classes, Objects

It can create a prototype object that holds functions and variables or a factory function that returns a new object.

OOP2: Classes, Objects, Encapsulation

Python's direct access convention prioritizes simplicity, readability, and flexibility. However, in scenarios where preventing unintended mutations of member variables or enforcing rules is necessary, using getters and setters can enhance encapsulation and maintain data integrity.

OOP3: Interfaces and Types

Since the `IShape` class is an abstract class that contains at least one pure virtual function, the `IShape` type cannot be used to instantiate concrete objects like `x` directly; it can only be used for pointers or references like ptr.

OOP4: Classes, Getters, Setters

The `protected` keyword in Java allows access to classes within the same package, even if they are not subclasses. This increases package cohesion and facilitates package-level collaboration. However, it also reduces encapsulation and makes access less predictable.

OOP5: Classes, Interfaces in Dynamically-typed Languages

Dynamically typed language interfaces serve as a conceptual contract that has no enforcement. On the other hand, statically typed language interfaces enforce a formal compile-time contract, ensuring that a class implements all required methods before the program is compiled.

OOP6: Subclass Inheritance, Interface Inheritance

- Interface Inheritance: in case the classes share the behaviors but the implementation are independent. For example, Playable interface > VideoPlayer, AudioPlayer, DVDPlayer, etc. They all share play and stop features, but the implementation are different.
- Subclass Inheritance: in case the classes share the same basic structure or functionality. For example, College > UCCollege > UCLA class. Every College has attributes like schoolCode and address, while UCCollege introduces additional shared features unique to the University of California system.

Soyeon Kim, 106292803

OOP7: Interface Inheritance, Supertypes and Subtypes

Part A

Interface A does not have supertype
Interface B has supertype of A
Interface C has supertype of A
Class D has supertypes of B, C, and A
Class E has supertypes of C, and A
Class F has supertypes of D, B, C, and A
Class G has supertypes of B, and A

Part B

Class D, F, and G can be passed to function `foo`

Part C

The `bletch` function cannot call `bar` in a statically typed language because A is the supertype of C, and `bar` may call functions or access members that exist in C but not in A. Since the compiler cannot guarantee that an A object is also a C object, the call is disallowed. To call `bar`, the parameter a must be explicitly and dynamically cast to type C.