OOAD/CSCI 448

Prof. Jim Dykes

20 Sep 2023

Daniel Hernandez

Project 2.1

1.

Design by Contract is a software design approach, often associated with OOP, that focuses on defining and enforcing precise agreements or contracts between different parts (classes, methods, or modules) of a software system. These contracts specify the responsibilities and expectations of each part to ensure that the system functions correctly. There are two main types of contracts in DbC: implicit contracts and explicit contracts. Implicit contracts are not explicitly defined in code but are assumed based on the typical behavior of the components involved. They rely on established conventions and expectations within the software development community. Explicit contracts are formally defined through comments, documentation, or language-specific annotations. They provide a clear and unambiguous specification of the expected behavior, inputs, and outputs of a component. Violating explicit contracts should result in an error or exception, assuming exception handling is built in.

2.

Default methods. This allows interfaces to have concrete methods, providing a default behavior that implementing classes can use or override. This feature was introduced in Java 8 and is particularly useful for evolving interfaces without breaking existing implementations

Static methods. Java interfaces can include static methods, which are not tied to any specific instance of a class. Static methods in interfaces can provide utility functions or factory methods related to the interface's concept.

Lastly, private methods. Modern Java interfaces can have private methods, which are used for code reuse within the interface itself. Private methods are not part of the interface contract and cannot be accessed by implementing classes.

3.

Abstraction:

Abstraction is the process of simplifying complex reality by modeling classes based on the essential properties and behaviors that are relevant to the problem domain. In other words, it involves focusing on the high-level view of an object or system while ignoring unnecessary details. Abstraction allows you to create a model that captures the essential features of an entity without specifying all the implementation details.

Pseudo-Code:

```
// Define an abstract class representing a Vehicle
abstract class Vehicle {
    abstract void start();
    abstract void stop();
    abstract void accelerate();
}

// Concrete classes that extend Vehicle
class Car extends Vehicle {
```

```
    // Implementation specific to cars
}

class Bicycle extends Vehicle {
    // Implementation specific to bicycles
}
```

Encapsulation:

Encapsulation is a mechanism that bundles data (attributes) and the methods (functions) that operate on that data into a single unit known as a class. It restricts direct access to some of an object's components and prevents the accidental modification of internal states from outside the class. Encapsulation enforces data hiding, which means that the internal representation of an object is hidden from external code, and access to that data is controlled through well-defined interfaces (methods).

Pseudo-Code:

```
class BankAccount {
    private double balance; // Private attribute

    // Constructor
    BankAccount(double initialBalance) {
        balance = initialBalance;
    }
```

```
    // Public methods to interact with the balance
    void deposit(double amount) {
        // Implement logic to deposit money
        balance += amount;
    }

    void withdraw(double amount) {
        // Implement logic to withdraw money
        if (amount <= balance) {
            balance -= amount;
        } else {
            // Handle insufficient funds
            print("Insufficient funds");
        }
    }

    double getBalance() {
        // Provide read-only access to the balance
        return balance;
    }
}
```

Their Relationship:

Abstraction is a concept that focuses on the high-level view and modeling of objects, whereas encapsulation is a mechanism that enforces data hiding and bundles data and methods

together within a class. Abstraction helps in defining what an object should do, while encapsulation defines how an object should do it while hiding the internal implementation. Encapsulation is often used to implement abstraction. It allows you to hide the internal details of an abstract data type (class) while exposing a well-defined interface (abstract methods) that defines the abstraction.