ROBUST -

In addition, we want software to be ROBUST, that is, capable of handling unexpected inputs that are not explicitly defined for its application. For example, if a program is expecting a positive integer (perhaps representing the price of an item) and instead is given a negative integer, then the program should be able to recover gracefully from this error.

ADAPTABILITY -

Software, therefore, needs to be able to evolve over time in response to changing conditions in its environment. Thus, another important goal of quality software is that it achieves adaptability

REUSABILITY -

software be reusable, that is, the same code should be usable as a component of different systems in various applications.

OO DESIGN PRINCIPLES

-Abstraction

-Encapsulation

-Modularity

ABSTRACTION

-is to distill a complicated system down to its most fundamental parts. Typically, describing the parts of a system involves naming them and explaining their functionality.

-An ADT is a mathematical model of a data structure that specifies the type of data stored, the operations supported on them, and the types of parameters of the operations

ENCAPSULATION

-different components of a software system should not reveal the internal details of their respective implementations.

MODULARITY

-refers to an organizing principle in which different components of a software system are divided into separate functional units. Robustness is greatly increased because it is easier to test and debug separate components before they are integrated into a larger software system.

INHERITANCE

-In object-oriented programming, the mechanism for a modular and hierarchical organization is a technique known as inheritance. This

allows a new class to be defined based upon an existing class as the starting point.

-The subclass can differentiate itself from its superclass in two ways. It may AUGMENT the superclass by adding new fields and new methods. It may also SPECIALIZE existing behaviors by providing a new implementation that overrides an existing method.

POLYMORPHISM

-The word polymorphism literally means “many forms.” In the context of objectoriented design, it refers to the ability of a reference variable to take different forms.

-CreditCard card = new PredatoryCreditCard(...); // parameters omitted

This is a demonstration of what is known as the Liskov Substitution Principle, which states that a variable (or parameter) with a declared type can be assigned an instance from any direct or indirect subclass of that type. Informally, this is a manifestation of the “is a” relationship modeled by inheritance, as a predatory credit card is a credit card (but a credit card is not necessarily predatory).

DYNAMIC DISPATCH

- to know what method to call if a method is overriden in its subclass

SPECIALIZED METHOD

-an added method in subclass cannot be called if:

CreditCard card = new PredatoryCreditCard(...); // card.processMonth cannot be called

APPLICATION PROGRAMMING INTERFACE (API) or

INTERFACE

-In order for two objects to interact, they must “know” about the various messages that each will accept, that is, the methods each object supports. To enforce this “knowledge,” the object-oriented design paradigm asks that classes specify the application programming interface (API), or simply interface, that their objects present to other objects.

-This requirement is known as STRONG TYPING.

-Overriding is REQUIRED

-MIXIN classes is implementing two or more interface to define a new interface

ABSTRACT

-same with interface but can implement method signatures

-TEMPLATE METHOD PATTERN, in which an abstract base class provides a concrete behavior that relies upon calls to other abstract behaviors

CASTING

-CreditCard card = new PredatoryCreditCard(...); // widening

PredatoryCreditCard pc = (PredatoryCreditCard) card; // narrowing

-Number n;

Integer i;

n = new Integer(3);

i = (Integer) n; // This is legal

n = new Double(3.1415);

i = (Integer)

-Number n;

Integer i;

n = new Integer(3);

if (n instanceof Integer)

i = (Integer) n; // This is legal

n = new Double(3.1415);

if (n instanceof Integer)

i = (Integer) n; // This will not be attempted

**Object Oriented Programming**

* Computer programming paradigm based on the concept of “Object”
* Is a computer programming model that organizes software design around the objects rather than functions (functional programming)
* A software design approach in which concept of Object occupies the first place

**What is an Object?**

* “Everything is an Object”
* Is a virtual entity with specific list of properties which can distinguish if from other objects and behavior which allow to manipulate with these properties
* Data + Behavior = Object

**What is a Class?**

* Class is a template for objects
* Object is always an instance of a Class

**Functional Programming vs OOP**

* In OOP our main concept is **object**
  + Works best when the execution of the program is in structured way
* In Functional Programming our main concept is **function**
  + Works best when you don't care about order of functions execution

**OOP Advantages**

* **Modularity**
  + Structure
  + Loose Coupling and High Cohesion - Maintenance is easier
* **Scalability**
  + Scalable easily and predictable
* **Lower Cost of Development (reusability)**
  + Reusing code, reduce number of defects and centralizing functions
* **Security and Reliability**
  + Hide details of implementations from your Object user, robust and independent modules

*Programming is a question of trade-offs between maintainability, scalability, speed of development, code quality and reliability*