Object Orientation

by Chuck England

Types, Classes, and Objects Oh My!

Topics

- Introduction
- History
- Benefits
- Basic Concepts

What is Object Orientation?

- **Motivation**
  - **Find a better way!**
- **Reliability**
- **Reduce Complexity (Reduce Codebase)**
- **Reduce Coupling**
- **Ease Maintenance (Refactoring)**

We need to understand the history…

Old world

- **Long sequence of imperative commands**
- **Little work accomplished**
- **Lots of error checking (but never enough)**
- **Not very robust**
  - ***(Remember the error checking? We missed some)***

New world

- **Short concise**
- **Lots of work for each instruction**
- **Little or no error checking**
- ***Check only what you know how to handle***
- **Robust code**
  - ***(Based on things you have already tested)***

Simple Example

Old world example:

- **Description**

  Add two numbers

- **Code**

```
LDA 0x076C
ADD 0x004B
STA $A
```

New world example

- **Description:**

  Open a file, read its contents into a string, and close the file

- **Code**

```
string fileContents = File.ReadToEnd("myFile.txt");
```

Definitions

- **Coupling**
  - **Degree of dependence on other software**
- **Cohesion**
  - **How strongly related or focused are the responsibilities of a single class, or set of classes**
- **High cohesion == Low Coupling**

- **Coupling is required to make software useful**
- **BUT! Coupling is bad!**
  - **When code changes it affects all other code it touches**
  - **Strive for weak (loose) coupling**

What does it all mean?

The journey is about the duality of coupling…

- **No coupling, no reuse**
- **Too much coupling leads to**
  - ***Brittle software (hard to change)***
  - ***No growth (Can't evolve code)***
  - ***Every program is like starting over***

The Whirlwind Tour to "The History of Programming"

- Imperative command (Classic Recipe)
  - Sub-routines
- "Struct"ured Programming
  - Abstract Data Types (ADT)
  - Top Down Design
- Object Oriented Programming (OOP)
  - Classes
  - Inheritance
  - Polymorphism
  - Abstraction
- Object Oriented Design (OOD)
  - Bottom Up (Compositional)
  - UML

Extensions of OOP/OOD

- **COM/DCOM/COM+**
  - **Components**
    - ***Binary compatibility***
  - **Interface Programming**
    - ***Decouples users from implementation***
- **.Net**
  - **COM++? (Next evolution of COM)**
    - ***Uses data hiding to hide complexities of COM***
  - **Includes the .Net Framework**
  - **Introduction of C#**
- **Service Oriented Architecture (SOA)**
  - **Services**
  - ***Promote further decoupling***
  - ***Another plane for abstraction***
  - **Promotes re-use at a macro-level (service) instead of micro-level (class)**
  - **OOD/OOP still apply underneath the covers**

Extension of OOP/OOD

- Declarative Programming
  - Change code into configuration
  - Do not need to recompile to change the program
- Domain Specific Languages
  - Language Oriented Programming
  - Allows us to more easily define programming in terms of the problem domain
- Aspect Oriented Programming
  - Allows us to implement cross-cutting concerns with ease
    - Logging
    - Transactions

How does object orientation help?

- Organization
  - Groups “like” functionality
- Abstraction
  - Reduction of complexity
    - *Classes are used to define abstraction*
    - *Continuation of abstract data types*
    - *Classes define the*
      - attributes (properties)
      - behaviors (methods)
    - *Simplifies complex problems*
      - via levels of abstraction
  - Human brain can only handle 7 (+/- 2)
    - *Anything greater than this is complex*
- Encapsulation
  - Data hiding
  - Interface is well defined

How does object orientation help?

- Inheritance
  - Create organized hierarchies
  - “Is a” vs. “Has a”
- Polymorphism
  - Objects respond dynamically
  - Each has their own inherent behavior
- Code reuse
  - Better than copy and past
  - Better than libraries
  - Code can be used over and over
  - Fixing a defect in a single class fixes all users
- Extensibility
  - Existing code may be extended
    - *Provides new functionality*
    - *Existing software still works*
  - Make enhancements or changes
    - *With little or no impact on existing software*
    - *Few or no major changes to existing design*
  - Extends software life
    - *With evolution and refactoring*
      - Changes the perception of how software is built
  - Leverage the development of others

How does object orientation help?

- **Robustness**
  - **Code resides with its data**
  - **Code exists in a single place (less to go wrong)**
  - **Build on top of working code**
  - **Less code == Less complexity**
    - ***Which means greater reliability***
    - ***Fewer moving parts***
- **Productivity**
  - **Code reuse**
  - **Better Organization**
- **Type Checking**
  - **Can be checked for correctness at compile time**
  - **Less code required, since no checking at runtime**
  - **Checking at runtime is complex**
    - ***And you can't get it right***
    - ***Always something you did not think about***
- **Software Patterns**
  - **The "Gang of Four" Design Patterns**

    Elevate the level of design
    - ***Singleton***
    - ***Factory***
    - ***Adapter***
    - ***Bridge***
    - ***Builder***
    - ***Etc., etc., etc.***
  - **Design patterns and anti-patterns help provide cookie cutter implementations**
  - **Another level of reuse**
  - **Allows developers to communicate**

UML (Unified Modeling Language)

- A set of diagrams that help us to model software
  - Reduces coding effort by modeling before we code
  - Helps to deliver precise instructions
    - How something is to be built
    - Document how something was built
  - Use Case Diagrams
    - Define how software will be used
  - Class Diagrams / Object Diagrams
    - Defines how software artifacts are built
      - *Organization*
      - *Communication (messages)*
      - *Hierarchies*
  - Sequence Diagrams
    - Define a sequence of events
    - Show life-cycle of objects
    - Further defines use case
  - Activity Diagrams
    - Similar to flow charts
    - Describes step-by-step operations
  - State Diagrams
    - Describes the states of objects in a state system
  - (And there are more…)

Basic Concepts

- **Class**
  - **Defines the abstract characteristics of a thing**
  - **It defines its**

    attributes (properties)

    behaviors (methods)
  - **Example: Dog**

- **Object**
  - **Defines a specific instance of a class**
  - **We create an object by using the keyword "new"**
  - **Example: Lassie : Dog**