Maynooth University
National University of Ireland Maynooth

CS264 – Software Design

Introduction to Object Oriented Programming

Joe Duffin
Eolas Building
Room 1.45 – joseph.duffin@nuim.ie

Modularity

What are some of the principles that characterise good code design?

Possibly the most important principle is modularity

- Some principles of modularity:
  - Decomposability
  - Composability
  - Continuity
  - Information hiding
  - The open-closed principle
  - The single choice principle

- Decomposibility: decompose complex problems into subproblems
- Composibility: production of software elements that may be freely combined with each other to produce new software.
- Continuity: small changes in specifications yield small changes in architecture.

Information Hiding

- **Underlying question: how does one "advertise" the capabilities of a module?**
- Every module should be known to the outside world through an official, **"public" interface**.
- The rest of the module's properties comprises its **"secrets".**
- It should be impossible to access the secrets from the outside.
- A central objective of software design is the **separation of interface and implementation.**

The Information Hiding Principle

- The designer of every module must select a subset of the module's properties as the official information about the module, to be made available to authors of client modules.

- **Justifications:**
  - Continuity
  - Decomposability

The Open-Closed Principle

OCP: Design principle due to Bertrand Meyer.

Modules should be open and closed.

- Open module: May be extended.
- Closed module: Not modifiable.

The rationales are complementary:

- For closing a module: changes to an existing tested/proven implementation has high cost.
- For keeping modules open: allows modules functionality to be *inherited* and *extended* by subclasses.

The Single Choice Principle

- Whenever a software system must support a set of alternatives, one and only one module in the system should know their exhaustive list.

- Editor: set of commands (insert, delete etc.)
- Graphics system: set of figure types (rectangle, circle etc.)

- Compiler: set of language constructs (instruction, loop, expression etc.)

Encapsulation languages

- Basic idea: gather a group of routines serving a related oo-purpose, such as ***has***, ***insert***, ***remove*** etc., together with the appropriate data structure descriptions.

- Advantages:
  - For the supplier: Get everything under one roof. Simplifies configuration management, change of implementation, addition of new primitives.

  - For the client: Find everything at one place. Simplifies search for existing routines, requests for extensions.

Why use objects?

- Reusability: Need to reuse whole data structures, not just operations

- Extendibility, Continuity: Objects remain stable over time.

Object technology: A first definition

- Object-oriented software construction is the approach to system structuring that bases the architecture of software systems on the types of objects they manipulate — not on “the” function they achieve.

Issues of object-oriented design

- How to find the object types?
- How to describe the object types?
- How to describe the relations and commonalities between object types?
- How to use object types to structure programs?

Description of objects

- Consider not a single object but a type of objects with similar properties.
- Define each type of objects not by the objects' physical representation but by their behavior: the services (FEATURES) they offer to the rest of the world.

- External, not internal view i.e. ABSTRACT DATA TYPES

Object technology: More precise definition

- Object-oriented software construction is the construction of software systems as structured collections of (possibly partial) abstract data type implementations.

Classes: The fundamental structure

- Merging of the notions of module and type:
  - Module = Unit of decomposition: set of services
  - Type = Description of a set of run-time objects
    ("instances" of the type)

- The connection:
  - The services offered by the class, viewed as a module, are the operations available on the instances of the class, viewed as a type.

Classes: The fundamental structure

- From the module viewpoint:
  - Set of available services (“features”).
  - Information hiding.
  - Classes may be clients of each other.

- From the type viewpoint:
  - Describes a set of run-time objects (the instances of the class).
  - Used to declare entities (≈ variables), e.g.

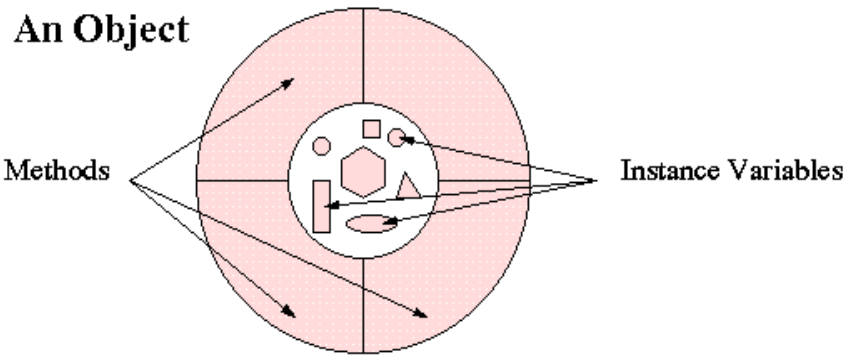
    ```
    C x;
    ```

  - Possible type checking.
  - Notion of subtype.

What is an Object?

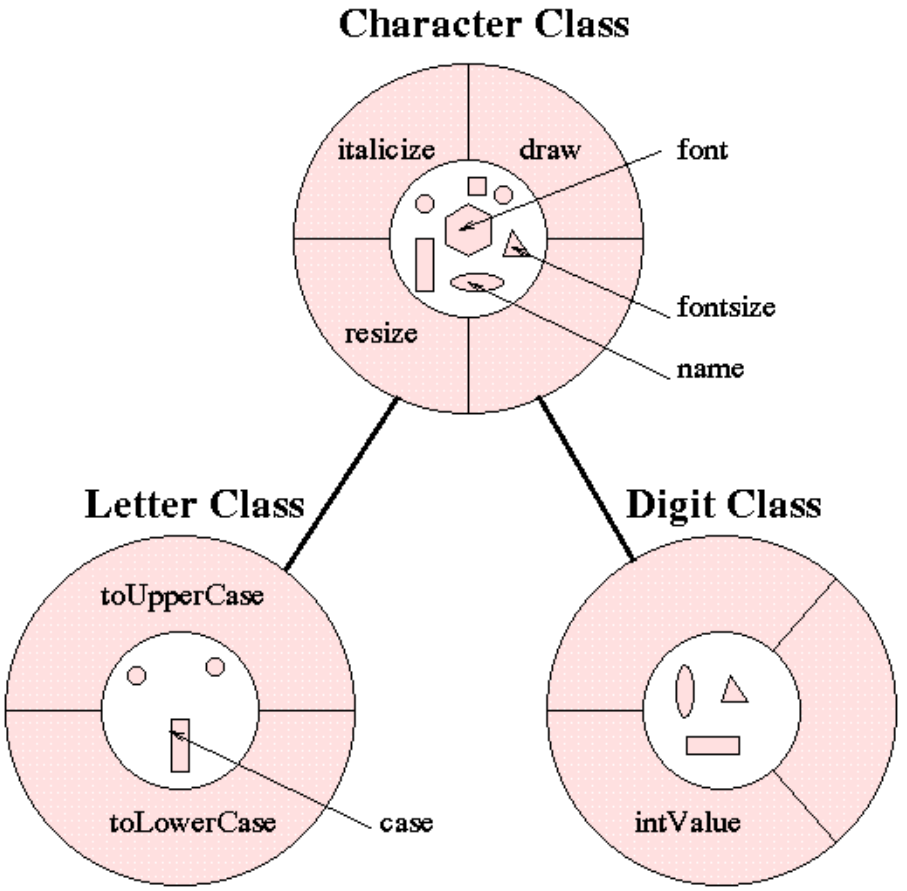
- An object has **state**, **behaviour** and **identity**.
  - **State**: an object can have internal data
  - **Behaviour**: methods
  - **Identity**: each object can be uniquely distinguished from every other (i.e. each object has a unique address in memory)

- Benefits of Using Objects:
  - Modularity
  - Information-hiding
  - Code re-use



What is a Class?

- A class is the blueprint from which individual objects are created.
- Describes a set of objects that have identical characteristics (data elements) and behaviours (functionality).
- An Object is an "***Instance***" of a Class



What is an Interface?

- Interfaces form a **contract** between the class and the outside world, and this contract is enforced at build time by the compiler.

- Implementing an interface allows a class to become more formal about the behaviour it promises to provide.

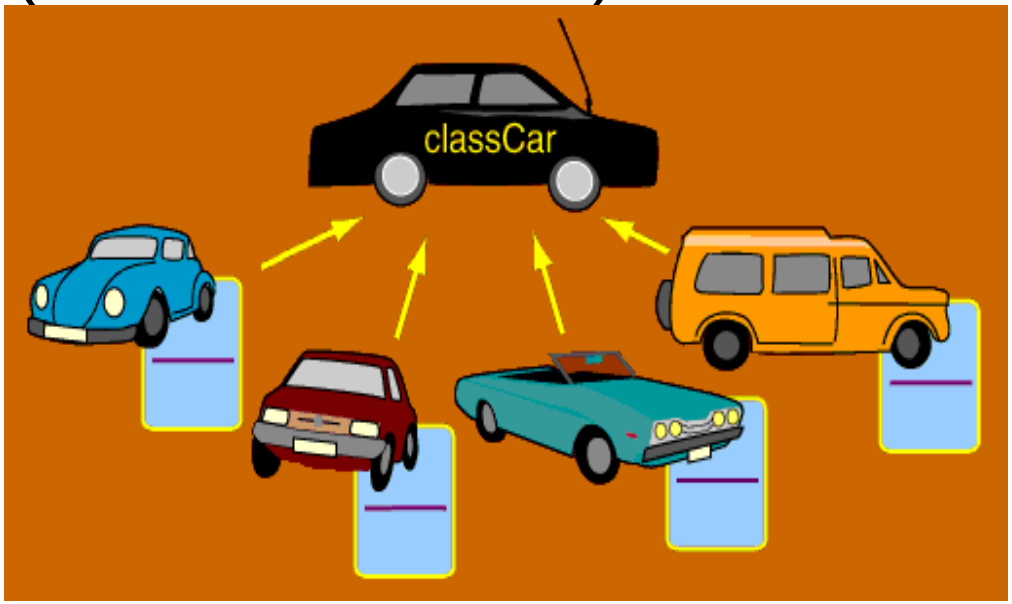
- If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

Object Oriented Design – key concepts

- **Abstraction**
- **Inheritance**
- **Polymorphism**
- **Encapsulation**

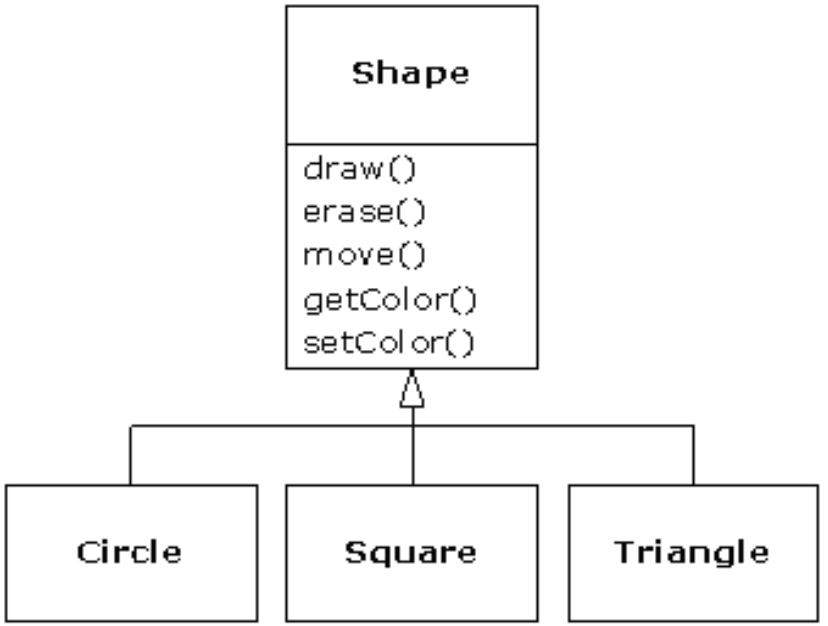
What is Abstraction?

- The identification of common properties and behaviours to form a new type
- Hence in OOP, types capture both properties and behaviours
- More importantly we can define new OOP types as we solve new problems.
- This approach of modelling the problem domain directly is central to OOD (and to its success)



What is Inheritance?

- Object-oriented programming allows classes to ***inherit*** commonly used state and behaviour from other classes
- Different kinds of objects often have a certain amount **in common** with each other.
- This allows you to create new objects that share base functionality with other objects, while implementing their own new, additional functionality.
- This supports to open/closed principle



What is Polymorphism?

- Treating an object not as the specific type that it is but instead as its base type.
- For example, we can treat both a circle as a shape and a square as a shape
- All shapes can be ”moved”. So to move a circle, we treat it as a shape and move it.

- **Advantage:**
  - Allows you to write code that doesn’t depend on specific types
  - Newly derived types can be added without making global changes

What is Encapsulation?

Encapsulation provides the basis for modularity- by hiding information from unwanted outside access and attaching that information to only methods that need access to it.

- **Public** means that anyone can access it.
- **Protected** means only inheriting classes can access it.
- **Private** means that only the class itself can access it.
- By default everything is private.