Cosc 241
Programming and Problem Solving
Lecture 23 & 24 (20/5/2019 & 23/5/2019)
Object oriented programming I & II

Lech Szymanski
lechszym@cs.otago.ac.nz

Keywords: abstraction, encapsulation, visibility, inheritance, polymorphism

UNIVERSITY of OTAGO
Te Whare Wānanga o Otāgo
NEW ZEALAND

Procedural vs. OOP

Procedural programming:

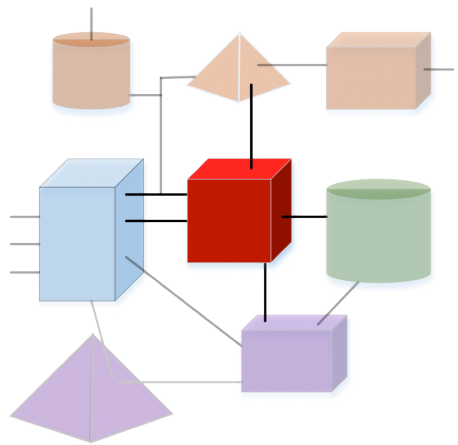
- functions act on data;
- a program organises function calls to manipulate data.

Object oriented programming:

- **objects** contain encapsulated data and associated methods;
- a program describes how objects interact via *messages*.

OOP

- Objects in real world vs. objects in programming: state (instance variables), behaviour (methods).
- Abstraction: hide the details of the implementation expose the interface; facilitate interchangeability.



- Encapsulation: bundle data and methods into one unit; hide the state of the object.
- Visibility: private/protected/public.

State

The state of an object is defined by:

- **instance variables** contain the state that is specific to a given instance of an object;
- **class/static variables** contain the state that is shared between all instances of an object; can be used without creating an instance of a given class.

Initialisation of the state is typicaly done via:

- default constructor,
- parametrised constructor,
- designated initialiser.

Behaviour

Methods are class specific functions that define what the object does and how it does it:

- **instance methods** – invoked on an object instance; can access instance variables for read/write;
- **class/static methods** – can be used without creating an instance of the class; cannot access instance variables

Inheritance

Inheritance is the creation of a subclass from a previously existing class; it allows re-use of code:

- inheriting parent methods,
- adding new methods,
- modifying, or overriding, existing methods.

Good reasons for using inheritance are:

- *specialisation* – subclass is a more specialised form of its parent;
- *specification* – subclass implements behaviour described, but not implemented, by its parent;
- *extension* – subclass provides new behaviour and capabilities.

Not the best reasons for using inheritance are:

- *limitation* – subclass restricts behaviour of the parent class;
- *generalisation* – subclass modifies behaviour of the parent to create a more general kind of object.

Inheritance vs composition

Composition is where a class includes another class as its instance variable:

- inheritance test: "is a" relationship;
- composition test: "has a" relationship.

Polymorphism

Polymorphism in OOP has to do with the same interface providing different functionality:

- method overloading – same method, different arguments
- method overriding – same method different behaviour, depending on

Interfaces and abstract classes

- An **abstract class** defines abstract methods – methods for which signature is given, but no implementation is provided.
- In Java an **interface** is in essence an abstract class.

Upcasting and downcasting

Casting refers to treating objects as if they were of different types.

Upcasting changes the type of an object to that of its parent class:

- implicit cast – can never fail because child object IS also its parent object;
- methods added by the child are not available after the upcast
- parent's methods that have been overridden by the child retain the overriden behaviour.

Downcasting changes type type of an object to that of its child class:

- explicit cast – can fail because a given object may or may not be an instance of the expected subclass;
- usually done to reverse upcasting;
- generally considered a bad practice.