Object-Oriented Programming

- Major components and issues:
  - Inheritance
  - Instance variables/methods vs. class variables/methods
  - Single vs. multiple inheritance
  - Dynamic binding/dynamic dispatch/polymorphism
  - Abstract classes

Some Terminologies

- Messages: call to methods
- Message protocol/interface: collection of methods
- Message passing: calling a method

Exclusivity of Objects

- Are all types objects? Are there primitive types?
- Advantage: Uniformity in language and its use
- Disadvantage: even simple operations must be done through message-passing process (e.g. adding two integers), can be less efficient
- Common: retain primitive types from imperative languages, add object-oriented support

Subclasses vs. Subtypes

- Principle of Substitution: A variable of a class can be substituted for a variable of one of its ancestor classes in any situation, without causing type errors and without changing the behaviour of the program
- If class B is a subclass of class A, and the behaviour of the object of class B is identical to that of object of class A when used as an object of class A, then B is a subtype of A.
- e.g. In Ada: `subtype Small_Int is Integer range -100..100;`
- For subtypes to work, inheritance must be public.
- Not all subclasses are subtypes, and not all subtypes are subclasses
- Subclasses are by default subtypes in many languages (C++, Java) unless methods are overriden.

Single vs Multiple Inheritance

- Multiple inheritance: allows inheritance from more than one class
- Can be useful
- Can be ambiguous, especially with diamond inheritance
- Languages that support multiple inheritance often have ways to specify diamond inheritance (virtual inheritance in C++)
- Java: multiple inheritance only on interfaces

Allocation and Deallocation of Objects

- Can they be allocated on the stack? Or must they be a reference/pointer to objects on the heap?
- Stack dynamic: what if an object of class B is assigned to an object of class A?
- In C++, this results in object slicing and lose data. Need to use pointers explicitly to avoid this.
- In Java, there is no issue with losing data

Dynamic vs. Static Binding

- Dynamic binding: if a variable can hold an object of class A or objects of any subclass of A, then the version of the method called on the object should depend on the real class of that object.
- Static binding: the method called is based on the (static) type of the variable referring to the object.
- In some languages, dynamic binding is done (e.g. Java)
- Some languages allow users to choose. Why? (e.g. C++)

Nested Classes

- Many languages allow classes to be defined inside other classes
- Visibility is limited, different languages have different rules

Smalltalk

- Perhaps the first object-oriented language
- Everything is an object, even integer constants
- No nested classes or multiple inheritance
- Even adding two numbers is implemented as sending a “+” message to one of the operands
- All objects are allocated from heap and referenced through reference variables
- Only dynamic binding supported, dynamic type binding

C++

- Objects on top of primitive types
- Both imperative and object-oriented
- Objects can be static, stack dynamic or heap dynamic
- Multiple inheritence, nested classes supported
- Static binding by default, dynamic binding can be specified
- public, private and protected members and inheritance
- pure virtual functions and abstract classes

Data Storage

- Class instance record (CIR): storage structure of instance variables of an object. Similar to a record
- Every class has its own CIR, known at compile time
- Subclasses have CIRs that are copies of those of parent class, with extra "fields" for additional instance variables

Dynamic Binding

- When dynamic binding is used, the CIR for each class needs to have information about the methods it defines
- Typically address/pointer to the code for the methods
- A virtual method table (vtable) is used to hold the address to each method defined in the class
- A pointer to the vtable is stored in the CIR
- When a method is called, code is generated to look at the vtable entry and call the appropriate version of the method
- Multiple inheritance: possibly needs multiple pointers to multiple vtables.