Programming Concepts Using Java

Week 1 Revision

W01:L01: Introduction

- Explore concepts in programming languages
  - Object-oriented programming
  - Exception handling, concurrency, event-driven programming, ...
- Use Java as the illustrative language
  - Imperative, object-oriented
  - Incorporates almost all features of interest
- Discuss design decisions where relevant
  - Every language makes some compromises
- Understand and appreciate why there is a zoo of programming languages out there
- ... and why new ones are still being created

W01:L02: Types

- Types have many uses
  - Making sense of arbitrary bit sequences in memory
  - Organizing concepts in our code in a meaningful way
  - Helping compilers catch bugs early, optimize compiled code
- Some languages also support automatic type inference
  - Deduce the types of a variable statically, based on the context in which they are used
  - `x = 7` followed by `y = x + 15` implies `y` must be `int`
  - If the inferred type is consistent across the program, all is well

W01:L03: Memory Management

- Variables have **scope** and **lifetime**
  - Scope — whether the variable is available in the program
  - Lifetime — whether the storage is still allocated
- Activation records for functions are maintained as a stack
  - Control link points to previous activation record
  - Return value link tells where to store result
- Two ways to initialize parameters
  - Call by value
  - Call by reference
- Heap is used to store dynamically allocated data
  - Outlives activation record of function that created the storage
  - Need to be careful about deallocating heap storage
  - Explicit deallocation vs automatic garbage collection

W01:L04: Abstraction and Modularity

- Solving a complex task requires breaking it down into manageable components
  - Top down: refine the task into subtasks
  - Bottom up: combine simple building blocks
- Modular description of components
  - Interface and specification
  - Build prototype implementation to validate design
  - Reimplement the components independently, preserving interface and specification
- PL support for abstraction
  - Control flow: functions and procedures
  - Data: Abstract data types, object-oriented programming

W01:L05: OOPS

- Objects are like abstract datatypes
- Uniform way of encapsulating different combinations of data and functionality
- Distinguishing features of object-oriented programming
  - Abstraction
    - Public interface, private implementation, like ADTs
  - Subtyping
    - Hierarchy of types, compatibility of interfaces
  - Dynamic lookup
    - Choice of method implementation is determined at run-time
  - Inheritance
    - Reuse of implementations

W01:L06: Classes

- A class is a template describing the instance variables and methods for an abstract datatype
- An object is a concrete instance of a class
- We should separate the public interface from the private implementation
- Hierarchy of classes to implement subtyping and inheritance
- A language like Python has no mechanism to enforce privacy etc
  - Can illegally manipulate private instance variables
  - Can introduce inconsistencies between subtype and parent type
- Use strong declarations to enforce privacy, types
  - Do not rely on programmer discipline
  - Catch bugs early through type checking