Principles of Computer Science I

Prof. Nadeem Abdul Hamid

CSC 120A - Fall 2004

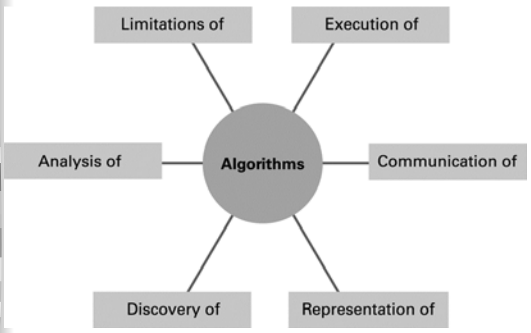
Lecture Unit 14

Review

Final Exam

- December 6 – 10:30 AM
- Can use
  - Textbook
  - One (double-sided) page (8.5 x 11″) of notes
- Can't use
  - Any other materials (lecture notes/lab work/etc)
  - Computer/calculator
- Thursday lab
  - Work on homework 9 problem
  - Ask questions, review for final
  - Course evaluations

Algorithms and Computer Science



Blind Men and the Elephant

Designing Algorithms/Programs

- Break large tasks into smaller, easier, more concrete subtasks (functional decomposition)
- Build abstractions
  - simplifications/elimination of irrelevant details
- Abstractions should be well-encapsulated
- Encapsulation
  - How an abstract is implemented "behind-the-scenes" should not affect other pieces of the design that use the abstraction
  - Abstraction should provide a well-defined interface

Object-Oriented Design Concepts

- View a task to be accomplished as a bunch of entities (objects/agents) interacting to solve the problem
- Programs are a collection of interacting objects that communicate with (call methods of/send messages to) each other
- Objects are a combination of data and associated operations that is supported upon that data

Classes and Objects

- A class describes the data (fields) and operations (methods) of its objects
  - Every object belongs to some class
- An object contains data (instance variables/fields) representing state, and instance methods, which are the things it can do
  - Class may also contain its own data (class variables/static fields) and class methods, denoted by the keyword "static"
- Classes form an inheritance hierarchy (tree), with Object at the root
  - Every class, except Object, has exactly one immediate superclass, which may be denoted by the keyword "extends"
  - Subclasses inherit all the (non-private) fields and methods of superclasses (except the constructors)

Constructors

- Class defines one or more constructors for initializing new objects of that class
  - If you don't provide a constructor, Java provides a default constructor with no arguments:
    - sets numeric/character fields to zero, booleans to false, object references to null
- Purpose of a constructor is to create an object in a valid state
- First thing a constructor does is call its superclass' constructor
  - If you don't explicitly do this, using "super(…)", then Java implicitly invokes (calls) the default constructor of the superclass
- Constructor for a class can call another constructor for the same class using "this(…)" as the first statement in the constructor
  - Avoids duplicated blocks of code in constructors

Using Objects and Casting

- Declare object variables just like other variables
  - <classname> <objectvar>;
- But an object is not instantiated (allocated space and initialized) until you create one using the "new" keyword to call the appropriate constructor
- An object can be assigned to a variable of its own class or any of its superclasses
  - In the other direction, *i.e.* to assign an object to a variable of a subclass, you have to use an explicit cast
  - Casting an object to a more general type is call upcasting and is always legal
  - Casting an object to a more specific type is called downcasting and Java will check at run-time if it is legal
  - Casting does not affect what the object *is*, only what fields and methods are available on the object at the position the cast occurs

Classes and Objects (cont.)

- The "instanceof" operator tests whether an object is an instance of a class
  - Returns true if the object is the class or any subclass thereof
  - Well-designed programs rarely use this
- A Java source code file may only contain one "public" class
  - Other non-public class definitions may be included in a single file
  - Name of the file must be the same as the name of the public class, but with a ".java" extension
- Classes should be as self-contained and independent as possible and reasonable
  - The interface (public fields and methods… made available to other code) should be kept small
- An object is responsible for keeping itself in a valid state at all times
  - Should limit access to its important data (fields)

Access

- Fields (instance/class variables) and methods are accessed by name
- Three dimensions to accessing name
  - Namespace
  - Scope
  - Access modifiers
- Java has six different namespaces:
  - Package names, type names, field names, method names, local variable names (including parameters), and labels
  - Identical names of different spaces do not conflict- *e.g.* a method may be named the same as a local variable – but it is best to avoid reusing names like this
- To refer to an instance feature (field/method) in a different object, use the syntax "otherObject.name"
- To refer to a class (static) feature in a different class, use the syntax "OtherClass.name"

Scope

- Scope of an identifier (name) is the part of a class/file where it is "visible" or legal to use
  - Variable declared anywhere in a class can be seen everywhere in a class
  - Scope of method's parameters is the entire method
  - Scope of a variable declared in a block (indicated by braces, { }) extends from the declaration to the closing brace
  - Scope of a variable declared in the initialization part of a for loop is the entire body of the loop
- Class variables and methods (indicated by "static") can be used anywhere within the class
- Instance variables (fields) and methods can be used anywhere except in static methods
- Within an instance method, the keyword this refers to the object on which the method is currently executing
  - When a field and a local variable have the same name, the name refers to the local varible; use the prefix this. to refer to the field

Access Privileges

- public: can be accessed from anywhere
- protected: can be accessed from any other class in the same package (folder) or from any subclass
- package (default): can be accessed from any other call in the same package
- private: cannot be accessed from outside the class
  - But private fields and methods can be accessed by other objects of the same class

Referring to Names

- Using fully qualified name:
  - java.io.BufferedReader inFile = …
- Or import a specific class or all (using *) classes from a given package at the top of the file and then just use the name
  - import java.io.*;
    …
    BufferedReader inFile = …

Methods

- A method is a named, executable chunk of code
  - All executable statements must be in methods (one or two exceptions, which we won't mention here)
- Method has a signature: name and number and types of its parameters
- Method has a return type (not part of its signature)
  - If the return type is other than void, the method must return a value of the specified type in every possible case
- Method may define local variables (scope, *etc.*)
  - Concepts of static/public/private/etc. do not apply to local variables
  - Local variables have undefined values until they are initialized
- Every method must have a unique signature within a class
  - Methods in other classes (including sub/superclasses) may have the same signatures

Executing Methods

- Executing a method means to cause its statements to be performed upon a given object
  - Also referred to as "calling a method upon an object", "invoking a method on an object", "sending a message to an object"
- Method invocation consists of
  - A reference to the object (often by name) or class
  - A dot
  - The name of the method
  - Zero or more "arguments" enclosed in parentheses
- When a method call occurs, the values of the arguments are copied into the corresponding parameters of the method
- Upon completion, the result of a method call expression is its return value

Polymorphism

- "Having many forms"
- Polymorphism in Java
  - Ability to assign objects to superclass variables
  - Overriding methods
- Overloading methods
  - When a single class declares two or more methods with the same name but different signatures
  - When a method call is made, the method with the best matching signature is used ("invoked")

Overriding Methods

- A class declares a method with the same signature as an inherited method
  - Return type of an overridden method must be the same too
  - Overriding method may not throw more exceptions than those thrown by the overridden method
- When the method is invoked on an object (or class), the overriding method is the one used, even if the object is being reference through a variable of a superclass
- Can still invoke the superclass' version of the method (from inside the class) using "super.<*name*>(<*parameters*>)"
- Shadowing or hiding refers to this same phenomenon in the context of fields