The University of Melbourne
Department of Computer Science and Software Engineering
433-254 Software Design
Semester 2, 2003

Answers for Tutorial 4
Week 5

1. Discuss parameter passing by value and by reference in Java with suitable examples. Compare them with parameter passing in C.

   Sample Answer:
   Passing by value (*call by value*): A copy of the parameter's value is passed to the called function. Therefore, subsequent changes to that copied value within the body of the called function will not affect the original value of the passed parameter.

   Passing by reference (*call by reference*): The caller allows the called function to change the original value of the passed parameter.

   In Java, if the passed parameter is an object, the value of a reference to that object will be passed to the method (*passing by value*). Parameters of type basic data types (e.g. int, float, double) are also passed by value. Thus, one can conclude that Java supports only call by value!

   In C, all calls are also call by value only (where call by reference is implemented by passing the value of a pointer)!

   See the example in lecture 2, week 5 (slides 7-12):

2. What is delegation? Consider an example of Student Class that delegates responsibility to a class called Person, which maintains person's details such as name, sex, and age.

   Sample Answer:
   In the context of OOP, delegation means that a class relies on other classes' services for implementing its own services (by calling methods upon objects of type other classes).

   Delegation usually happens when classes are in *containership* or *client-supplier* relationship. But, class Student and class Person are more likely to be an *is-a/inheritance* relation, which makes it bit tricky to have a good delegation example! Anyhow, here is an example:

```
class Person
{
    String          name;
    String          DOB;
    Character       sex;

    Person( String n, String date, char s)
    {
            name = n;
            DOB = date;
            sex = new Character(s);
    }
}

class Student extends Person
{
    String stud_number;
    Subject[] subjects;

    Student(String n, String dob, char s, String number)
    {
            // Delegating initialization of the name, DOB and sex to Person.
            super(n, dob, s);
            stud_number = number;
            subjects = new Subject[20];
    }

    public setResult(int i, float mark)
    {
            // Delegating setting of marks to Subject.
            subjects[i].setMarks(mark);
    }
}

class Subject
{
    String name;
    Float points;
    Float marks;

    Subject(String n, float f)
    {
            name = n;
            points = new Float(f);
    }

    public setMarks(float m)
    {
            marks = new Float[m];
    }
}
```

3. Discuss various forms of inheritance with example.

   Sample Answer:
   Single Inheritance
   Multiple Inheritance
   Multilevel Inheritance
   Multi-path Inheritance
   Hierarchical Inheritance
   Hybrid Inheritance

   See lecture 2, week 4 (slides 4-5).

4. Discuss syntax of defining inheritance, final classes, "super" construct, and "protected" variables. If possible, also discuss how Java handles shadow variables and overridden methods.

   Sample Answer:

   class *Subclass* **extends** *Super-class*

   A *final* class can NOT have subclasses (i.e. can not be sub-typed).

   A super-class constructor can be accessed within a subclass constructor using keyword *super*. In fact, any method or variable of a super-class (subject to their visibility) can be accessed by its subclasses using prefix *super*.

   Protected variables of a class can be accessed by all classes in the same package of that class and all subclasses of that class in any package. For more information on visibility rules (see Lecture 1, week 4, slides 17-19).

   Non-final methods of a class may be *overridden* by its subclasses. That is, a subclass may implement an inherited method differently. However, the overridden method has to maintain the signature of the original method (i.e. method's name, type of the return value as well as order and type of the parameters). Thus, any reference to such an overridden method within the body of its defining class (and its subclasses) will be resolved to the overridden version of the method and if for any reason the original version is needed to be called, that must be qualified with *super* prefix.

   A shadow variable is a variable in a subclass with the same name as an inherited variable from the super-class (similar to overridden methods). Thus, any reference to such variable within the body of the defining subclass (and its subclasses) will be resolved to the shadow version and if for any reason the original version is needed to be called, that must be qualified with *super* prefix.

5. Discuss differences between Arrays and Vectors in Java.

Sample Answer:
Array is a fixed size container for storing sequence of items of the same type, where each item can be accessed using its index in the sequence. However, Vector is a variable size container for storing items of different types. Items within a vector can also be accessed using an index.