

# Programming Language Concepts

Mid-Semester Examination, Semester IV

21st February, 2006

*Answer all questions, the paper is for 25 marks, 3 hrs.*

1. Suppose we want to write a program to display a panel with three buttons, labelled *Create Child*, *Create Sibling* and *Paint Descendants Red*.
  - Pressing *Create Child* creates one more panel with three buttons that behave exactly like the first one. The new panel is considered a child of the panel that created it.
  - Pressing *Create Sibling* creates one more panel with three buttons that behave exactly like the first one. The new panel is considered a sibling of the panel that created it, that is, the new panel has the same parent as the one that created it.
  - Pressing *Paint Descendants Red* makes all panels which are descendants of this panel paint their backgrounds red, where descendants mean children, children's children, ...

Write an (informal) description of how you would write such a program in Swing - use pseudocode and/or diagrams to describe how events are handled. (5 marks)

2. Assume that you have available a method

```
static int System.in.readPerfectSquare()  
    throws NotSquareException , EOFException
```

that reads a value of type `int` from the keyboard and returns the value read if it is a perfect square. This method generates a `NotSquareException` when a value that is not a perfect square is read and an `EOFException` when the end-of-file character (Ctrl-D in Unix) appears.

Write a Java loop to read upto 100 values of type `int` from the keyboard using the method `System.in.readPerfectSquare()` and print the sum of the values read. All `NotSquareExceptions` should be ignored. The length of the input sequence may be less than 100 and is not known in advance. The sequence is terminated by a Ctrl-D. Assume that all values typed before Ctrl-D are compatible with type `int`. (4 marks)

3. Assume that we have defined a parameterized class `BinaryTree <T>` in Java to represent a Binary Tree in which each node can hold a value over a uniform type `T`.

Consider the following code that uses Java's reflective features:

```
import java.lang.reflect.*;
...
BinaryTree <String> btree1 = new BinaryTree <String>();
BinaryTree <String> btree2 = btree1.getClass().newInstance();
```

Explain why the assignment to `btree2` via `newInstance()` in the second line generates a complaint about type incompatibility. (2 marks)

4. Here is a skeleton definition in Java of a class `Stack` that implements a generic stack as an array of `Object`.

```
public class Stack{
    private Object[] data = new Object[100];
    private int size;
    ...
    public boolean isEmpty(){...}
    public Object pop(){...}
    public void push(Object o){...}
}
```

Rewrite this skeleton code using type variables so that each instance of `Stack` can store any value that is a subtype of the concrete type supplied when instantiating `Stack`. (3 marks)

5. All the savings accounts of a bank that supports online banking are stored in a single object of class `BankAccount`. This class supports functions for balance enquiries and transfers between accounts in the bank. This corresponding functions have the following signature:

```
public double get_balance(int accno);
//return the contents of account accno

public boolean transfer(int src, int tgt, double d);
//transfers amount d from account src to account tgt
//return true iff successful
```

To access the account, a user must login with a valid username and password. After logging in, the user can perform *one* balance enquiry or transfer. To invoke another transaction, the user must log in again. There is no restriction on a user logging in multiple times in parallel. Each successful login in parallel can separately perform one transaction.

Logging into the account is implemented by a function of the form

```
public ... login(int accno , String u, String p);
//log in as user u with password p to access account accno
```

Suggest how we can implement the login function to provide the required limited access BankAccount. Describe your design in terms of Java-like pseudocode. Give details of data definition and function interfaces. For function bodies, you can write comments instead of detailed pseudocode to explain the features of your design. (5 marks)

6. Consider the followign class definitions in Java:

```
public abstract class Vehicle{
    public abstract int doors();
    public abstract int wheels();
    public boolean equals(Vehicle v){
        return (this.doors() == v.doors());
    }
}
public class Truck extends Vehicle{
    private boolean heavy;
    public Truck(boolean isheavy){
        heavy = isheavy;
    }
    public int doors(){return 2;}
    public int wheels(){
        if(heavy){return 6;}else{return 4;}
    }
    public boolean equals(Truck t){
        return (this.wheels() == t.wheels());
    }
}
public class Car extends Vehicle{
    private boolean sporty;
    public Car(boolean issporty){sporty = issporty;}
    public int doors(){
        if (sporty) {return 2;}else{return 4;}
    }
    public int wheels(){return 4;}
    public boolean equals(Vehicle v){
        return (this.wheels() == v.wheels());
    }
}
```

Explain each line of output produced by the following code.

```
public class TestVehicles{
```

```
public static void main(String[] args){
    Car porsche = new Car(true);
    Truck volvo = new Truck(true);
    Truck minidor = new Truck(false);
    Object o = porsche;
    Vehicle p = porsche;
    Vehicle v = volvo;

    System.out.println("1. "+ o.equals(volvo));
    System.out.println("2. "+ v.equals(minidor));
    System.out.println("3. "+ p.equals(v));
    System.out.println("4. "+ minidor.equals(porsche));
    System.out.println("5. "+ porsche.equals(v));
    System.out.println("6. "+ volvo.equals(minidor));
}
}
```

(6 marks)