Exam 2 Coverage and Review

Expectations:

(1) Know (the covered) language features well

(2) Know Java's support as well as your group's language for (the covered topics) features (yes or no, if yes, code example, if no why not).

(3) Know the various ways (general concepts only, not coding examples) shared by the groups for supporting control statements, subprograms, and OOP.

(4) Be able to use language evaluation criteria to discuss pros and cons of (the covered) features

Focus of each programming language feature/concept:

What? Why? How?

Coverage:

statements, subprograms, ADTs and OOPs, exception handling, and functional programming

Question format

- True/False, Multiple Choice, Fill in Blank
  - choose the best answer
- Short Answer
  - In case there's ambiguity or error, state that in your answer, give your assumption, and answer your question based on your assumption.
  - Illustration (code fragment, skeleton).
  - Justification /Explanation: use criteria, be objective.
  - Group activity related topics.
  - Open questions.

Detailed Topics

Statements

- Selections:
  - if: Boolean conditions, dangling else problem and the proposed solutions.
  - switch: cases (multiple values, sub ranges etc.), control flow (explicit break or not)
  - design issues for if and switch
- Iterations:
  - Counter controlled, logically controlled, pre-test, post-test
  - Compare and contrast, rewrite codes

Subprograms

- Parameter passing (by value, by reference, by assignment, modes – in, inout, out)
- Default, keywords/named, variable length parameters/arguments
- Advanced features – subprogram name as parameters, overloading, generic subprograms, lambda expressions (anonymous subprograms)

OOP design and features
- Writing ADT
- Parameterized ADT
- Inheritance: concepts and writing base/derived classes
- Overloading and overriding methods, operator overloading
- Polymorphism and static/dynamic binding
- Multiple Inheritance
- Abstract methods and abstract classes

Exception handling
- basic concepts
  - control flow including exception propagation, throwing/catching exceptions, …
- coding try-throw-catch: Java
- default handler
- Java vs. C++ in exception handling

Functional programming (*the following concepts are already covered in subprograms)
- Referential transparency
- Recursion and tail recursion
- Lambda expressions
- Higher-order functions (i.e. function names as parameters)

Group activity related questions:
- Trends in the design of the language constructs
- Compare and contrast of different designs (of the same feature/constructs)

Open questions
- Group's language in support of language features covered in this test
- Compare and contrast with Java
- Strength and weakness in design
- Your suggestion for improvement

Mock Test (for practice only, not reflecting exact coverage of the test)

Part 1: Statements and Subprograms

1. if statement (e.g. condition restricted to Boolean or numerical values or objects okay; dangling else problem)

Multiple Choice:

A: okay

B: Fail
C: compile-time error
D: Run-time error

Case (1): C++; Case (2): Java Case (3): change b1 and b2 to integers with b1 = 1, b2 = 0;
//b1 and b2 are two Boolean values, with b1=true, b2=false;
if (b2 = b1) { …print "okay"… } else { …print "fail"… }

2. switch statement (e.g. writing switch statement in Java; weakness in Java's switch statement; code conversion etc.)

Multiple Choice: Java-like code. initially x is 5, what will be x if myChoice is 1

```
switch (myChoice) {
   case 1:     x *= 3;
   case 2:     x += 2;
   default:    x = 0;
}
```

//now display x A: 15 B: 7 C: 17 D: 0

3. counter controlled and/or logically controlled loops (e.g. counter vs. logical; rewriting loops for <-> while <-> do-while; loop with break/continue)

Multiple Choice: Java-like code.

```
for (int j=0; j<7; j++) {
   print(j);   //assume that will display j's value
   if (j % 2 == 1) {  j++; }
   else  {
        if (j == 3) break;
   }
} //what will be printed out
```

A. 0, 1, 2, 3, 4, 5, 6 B. 2, 4, 6 C. 0, 1, 3, 5 D. 0, 2, 4, 6

Problem: Convert the following Python loop to Java loop, make sure they are semantically equivalent, i.e. print out the same values

```
for  j in range (100) :
        j = j * 2
        print(j)
```

Problem: Given the following Java-like code,

```
int ct = 0, sum = 0;
do {
        ct++;
        sum += ct;
} while (ct <= 100);
```

Convert to a for loop (in Java/C++). Only one for loop allowed, no additional statements could be used outside for loop.

4. subprogram concepts (default parameters, variable length parameters, …)

Problem: Describe a situation where ”variable length” parameters is useful; use code skeleton to illustrate the usage (e.g. function definition and function call)

5. parameter passing -- modes, models (pass by value, reference, assignment)

True or False:

(1) Viewing from subprogram implementation, Java only implements pass by value.

(2) In C# parameter passing, a keyword ‘out’ is used to indicate an “out mode” parameter.

Practice: Given a function fun with two integer parameters, par1 and par2, the body of function performs

```
par1  += 2
par2 = par1 * par2
```

first and second are two integer variables with first = 5, second = 2

If par1 is a value parameter and par2 is a reference parameter, what will be first and second value after the function call fun(first, second)

6. advanced subprogram features (e.g. overloading, subprogram as parameter, generic subprogram) – what & why? Code skeleton to explain

   True or False: Java support subprogram overloading.

   Problem/Discussion: A lot of contemporary programming languages support function as parameter

   (a) Describe a scenario (i.e. give an example) which shows such feature (i.e. function as parameter) is helpful/useful.

   (b) Use your favorite language (which supports function parameter) to write a header line of the subprogram as well as a call to the subprogram for the example in (a).

7. advanced subprogram features – language implementation & compare/contrast

   In above practice problem:

   Open question: Here’s a sample code showing how C++ supports generic subprograms:

```
template <typename T>
T myMax(T x, T y)
{return (x > y) ? x : y;}
```

   If you were assigned a task to improve this feature in C++, what modification you’d do? Why? Justify your answer.

Part 2: ADTs and OOP

Problem 1: True/False Basic Concepts

(1) Java is a pure object-oriented language. Or, everything in Java such as an integer 25 is an object (i.e. exclusivity of objects).
(2) C++ supports multiple inheritance.
(3) In Java a class MtBike may inherit from a class Bike using protected base inheritance.
(4) Class M is an abstract class, so we cannot create an object of class M.
(5) In Java, when binding to overriding methods, by default dynamic binding always used.

Problem 2: ADT

(a) ADT provides encapsulation and information hiding. For the following C++ class, please describe how encapsulation and information hiding provided.

```
class Rectangle {
    int width, height;
 public:
   Rectangle (int, int);
   int area ();
   int perimeter();
};
```

(b) Rewrite the above code in Java.

(c) In your opinion which code (C++ or Java) provides better information hiding? Justify your answer briefly.

Problem 3: Parameterized ADT

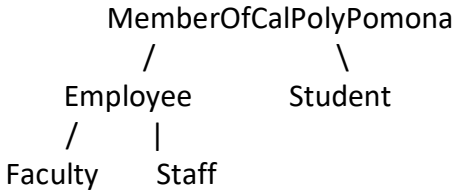
(a) What is a parameterized ADT? Describe briefly.

(b) Could Java ArrayList be considered as a parameterized ADT? Why or why not?

(c) How does C++ implement a parametrized ADT? (use code skeleton to illustrate.)

Problem 4: Inheritance

Assume we have the following inheritance hierarchy. Use this example to answer the following questions.



(a) Name one advantage of inheritance and use the above example to illustrate/justify the advantage you named.

(b) Write a code skeleton for the Employee class, assuming other classes are properly defined.

(c) What is multiple inheritance?

(d) Does the above inheritance hierarchy show an example of multiple inheritance?

Problem 5: ADT and Inheritance

A parent class Vehicle has two methods: void applyBrake (int decrement); void speedUp(int increment); however, you don't know how exactly to applyBrake or speedUp for a vehicle in general, i.e. not able to write the method body for these methods.

A subclass Auto inherits from Vehicle, and with the same two methods as the parent class but you know exactly how to applyBrake and speedUp in Auto.

A subclass Bike inherits from Vehicle, it has the speedUp as the Vehicle and you know how to speedup when riding a bike. For applyBrake, it has two methods implemented, void applyBrake (int decrement); and void applyBrake (Mode m) -- assuming Mode is a properly defined class.

A subclass MtBike inherits from Bike with the same methods as Bike for applyBrake with the speedUp method now taking a 2nd parameter void speedUp (int increment, Boolean flag), and it also has a setHeight feature void setHeight (int hgt);

(a) Draw the inheritance hierarchy for the classes described in this problem.

(b) Which of the above method (if any) should be declared as abstract method? Which of the above class (if any) should be declared as abstract class?

(c) Identify overloading methods (if any). (note: your answer should be in the form as "method X in class A overloads with method Y in class B".)

(d) Identify overriding methods (if any). (note: for your answer use similar format as above.)

Problem 6: Polymorphism
Given the following pseudo code description of two classes:

```
class   Animal:
        //constructors and destructors (if needed) are properly defined
        void cry () { … }
        void fur() { … }
        double weight( … ) { … }

class Dog:  inherits from class Animal
        //constructors etc. properly defined
        void cry() {…. }
        void fur() {… }
Animal myPets1 = new Animal();
Animal myPets2 = new Dog();
Dog d = new Dog();
```

(a) Identify polymorphic variables (also called polymorphic references) in above example.

(b) A method call, *myPets0.cry()* , under static binding, which method will it bind to? Under dynamic binding, which method will it bind to? (note: indicate "error" if such a call will result an error. Same for the following questions.)

(c) A method call, *d.cry()* , under static binding, which method will it bind to? Under dynamic binding, which method will it bind to?

(d) A method call, *myPets1.fur()* , under static binding, which method will it bind to? Under dynamic binding, which method will it bind to?

(e) A method call, *myPets[1].weight()* , under static binding, which method will it bind to? Under dynamic binding, which method will it bind to?

(f) Use C++ (or Java syntax) to write the above Animal and Dog class (no details about the methods needed) that indicates cry() and fur() methods are for dynamic binding while weight() method is for static binding.

Part 3: Exception Handling in Java

Problem 1: concepts

(a) What is the root class of all user-defined Java exception classes?

(b) If an exception occurs during the execution of a method m(), but m() doesn't catch that exception, what happens to that exception?

(c) For each of the exceptions below, identify it as checked or unchecked exception.

(1) NullPointerException

(2) FileNotFoundException

(3) ClassNotFoundException

(d) Name a strength in Java's design of exception handling as well as a weakness in Java's exception handling. Briefly justify your answer.

Problem 2: Programming with exception handling.

Use your favorite language to program the following cases with emphasis on exception handling (source code only). Open a file (if fail, raise an exception and handle it by printing a file not found message). Read in two integer values from the file (if not integer type value or don't have at least two values, raise exception and handle it by printing wrong data message), divide the first value by the second one (raise the exception and handle it by assigning 0.0 as the quotient).

Part 4: open question

Problem 1: Review language features (that within the discussed control statements, and subprograms) of your team's language. Name one feature that's well designed by your

language (name it, and show a code example). Name one feature that is weak in writability or readability.

Problem 2: OOP or Exception Handling feature of the team language.
Most modern languages (including our group languages) support OOP and exception handling features.
(a) Name your team language.
(b) Give an example (i.e. write code skeleton) to show how your team language support inheritance.
(c) Does your language support abstract class? If no, explain why it's not supported. If yes, show an example (i.e. write code skeleton) of an abstract class in that language.
(d) In terms of exception handling, name one difference of your team's language with that in C++. Which one (your group language or C++) has better design in term of the "difference" you stated here.