Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exercise 7

Dynamic Method Binding and Multiple Inheritance

November 6, 2020

Task 1

(from a previous exam)

Consider the following C++ program:

```
class X {
   public:
      X(int p) : fx(p) {}
      int fx;
};
class Y {
   public:
      Y(int p) : fy(p) {}
      int fy;
};
class B : public virtual X,public Y {
   public:
      B(int p) : X(p-1),Y(p-2){}
};
class C : public virtual X,public Y {
   public:
   C(int p) : X(p+1),Y(p+1){}
};
class D : public B, public C {
   public:
      D(int p) : X(p-1), B(p-2), C(p+1){}
};

int main() {
   D* d = new D(5);
   B* b = d;
   C* c = d;
   std::cout << b->fx << b->fy
             << c->fx << c->fy;
   return 0;
}
```

What is the output of running the program?

(a) `5555`

(b) `2177`

(c) `4147`

(d) `7177`

(e) `7777`

(f) None of the above

Task 2 (from a previous exam)

Consider the following C++ code (recall that default constructors, i.e., constructors without arguments, do not need to be called explicitly in C++):

```
class A {
    public:
        A(int i) { std::cout << "A" << i; }
        A() { std::cout << "A1"; }
        virtual int get() { ... }
};

class B: MODIFIER A {
    public:
        B(int i) : A(i) { std::cout << "B" << i; }
};

class C: MODIFIER A {
    public:
        C(int i) : A(i) { std::cout << "C" << i; }
};
class D: public B, public C {
    public:
        D(int i) : B(i + 10), C(i + 20) { std::cout << "D" << i; }
};
```

Now assume that `MODIFIER` is replaced by `public`.

A) Why does the following client code not compile?

```
void client()
{
    D* d = new D(5);
    std::cout << d->get();
}
```

B) Add a method to one of the classes so that `client` compiles.

C) What is the output resulting from the call `new D(5)` in method `client`?

D) Now, assume that `MODIFIER` is replaced by `public virtual`.

What is the new output resulting from the call `new D(5)` in method `client`?

Task 3

Consider the following C++ code:

```
class Person
{
   Person *spouse;
   string name;

public:
   Person (string n) { name = n; spouse = nullptr; }
```

```
   bool marry (Person *p)
   {
      if (p == this) return false;
      spouse = p;
      if (p) p->spouse = this;
      return true;
   }

   Person *getSpouse () { return spouse; }
   string getName () { return name; }
};
```

The method `marry` is supposed to ensure that a person cannot marry him-/herself. Without changing the code above, create a new object that belongs to a subclass of `Person` and marry it with itself.

Hint: use multiple inheritance. Explain what happens.

Task 4 (from a previous exam)

Consider the following Java classes:

```
class A {
   public void foo (Object o) { System.out.println("A"); }
}

class B {
   public void foo (String o) { System.out.println("B"); }
}

class C extends A {
   public void foo (String s) { System.out.println("C"); }
}

class D extends B {
   public void foo (Object o) { System.out.println("D"); }
}

class Main {
   public static void main(String[] args) {
      A a = new C(); a.foo("Java");
      C c = new C(); c.foo("Java");
      B b = new D(); b.foo("Java");
      D d = new D(); d.foo("Java");
   }
}
```

What is the output of the execution of the method `main` in class `Main`?

(a) The code will print A C B D

(b) The code will print A C B B

(c) The code will print C C B B

(d) The code will print C C B D

(e) None of the above

Task 5

Java 8 allows interface methods to have a default implementation directly in the interface.

A) What are some advantages of this feature?

B) What could be some problems with this feature? How can they be resolved?

C) What problems of C++ multiple inheritance are avoided by this new design for Java interfaces?

D) Now suppose that, in addition to method implementations, Java also allowed interfaces to define fields. Interfaces would not have constructors and interface fields would always be initialized with a default value.

- What are some advantages of this feature?
- Given the restrictions above, are there any problems left with such an implementation of multiple inheritance? If so what are they? Propose a solution for each problem you have identified.

Task 6

Consider the following C# classes:

```
public class Matrix {
   public virtual Matrix add(Matrix other) {
      Console.WriteLine("Matrix/Matrix");
      return null;
   }
}

public class SparseMatrix : Matrix {
   public virtual SparseMatrix add(SparseMatrix other) {
      Console.WriteLine("SparseMatrix/SparseMatrix");
      return null;
   }
}

public class MainClass {
   public static void Main(string[] args) {
      Matrix m = new Matrix();
      Matrix s = new SparseMatrix();
      add(m,m);
      add(m,s);
      add(s,m);
      add(s,s);
   }

   public static Matrix add(Matrix m1, Matrix m2) {
      return m1.add(m2);
   }
}
```

A) What is the output of this program? Please explain.

B) Without breaking modularity, change only the body of `MainClass.add` to make it possible to always call the most specific `add` method from the matrix hierarchy.

Task 7 (from a previous exam)

Consider the following C# code, which compiles and executes without raising exceptions:

```
1  class Ingredient {
2    public void mix(Ingredient i1, Ingredient i2) {
3      Console.WriteLine("Ingredient.mix");
4    }
5  }
6
7  class Milk: Ingredient {
8    public void mix(Egg e, Flour f) {
9      Console.WriteLine("Milk.mix");
10   }
11 }
12
13 class PowderedMilk: Milk {
14   public void mix(Ingredient i, Flour f) {
15     Console.WriteLine("PowderedMilk.mix");
16   }
17 }
18
19 class Egg: Ingredient {}
20
21 class Flour: Ingredient {}
22
23 class Program {
24   static void mix(Ingredient i1, Ingredient i2, Ingredient i3) {
25     (i1 as dynamic).mix(i2 as dynamic, i3 as dynamic);
26   }
27
28   static void Main() {
29     Ingredient i1 = new PowderedMilk();
30     Ingredient i2 = new Egg();
31     Ingredient i3 = new Flour();
32     mix(i1, i2, i3);
33   }
34 }
```

A) Which is the output of the execution of the method `Program.Main()`?

B) List **all** the casts (from line 25) and **all** the methods that can be removed from the given code, such that it still compiles and when executed produces the output from Task A.

Task 8 (from a previous exam)

Consider the following C++ class definitions, which compile fine:

```
class Box {
  private:
    int content;
  public:
    void set(int x) { content = x;     }
    int  get()      { return content; }
};

class IncreaseBox: virtual public Box {
  public:
    void increase() { set(1 + get()); }
};

class MultiplyBox: virtual public Box {
  public:
```

```
    void multiply(int x) { set(x * get()); }
};

class BestBox: virtual public IncreaseBox, virtual public MultiplyBox {};
```

In the lectures you have seen that in an object-oriented language one can simulate inheritance by a combination of subtyping and aggregation. In this task you will need to adapt this technique to handle multiple virtual inheritance in C++, and use it to translate the provided C++ code to Java, **without** using default methods in Java interfaces.

To help you check that the translation preserves the original behavior, we provide on the right an example of client code that (i) should compile fine in Java, and (ii) should compute the same values as the analogous C++ version on the left. Note that in the Java version it should be possible to create instances of `MultiplyBox` that are not instances of `IncreaseBox`, and vice versa.

```
// C++
void client(BestBox* bb) {
    Box* b = bb;
    IncreaseBox* ib = bb;
    MultiplyBox* mb = bb;

    b->set(10);
    ib->increase();
    mb->multiply(2);

    std::cout << b->get();
}
```

```
// Java
void client(BestBox bb) {
    Box b = bb;
    IncreaseBox ib = bb;
    MultiplyBox mb = bb;

    b.set(10);
    ib.increase();
    mb.multiply(2);

    System.out.print(bb.get());
}
```

Draw a graph of the translated Java code, expressing classes, interfaces and the relations between them. Each node must be a class or interface (write which one) with an intuitive name; the directed edges must be labeled with "implements", "extends" or "has-a", according to the relation that they express. You do not have to specify fields, methods nor constructors.