Module 3F6: Software Engineering and Design

OBJECT ORIENTED SOFTWARE DESIGN

Examples Paper 1

Straightforward questions are marked †
Tripos standard (but not necessarily Tripos length) questions are marked *

Object-oriented programming, UML

1. Examine the program shown Fig. 1 and answer the following questions:

   (a) † What output will be produced when the program runs?
   (b) † Draw a class diagram showing the structure of this program.
   (c) † Draw a sequence diagram showing what happens when this program runs.

```cpp
#include <iostream>
using namespace std;

class Base {
    public:
        virtual void print()=0;
        void say_hello();
};

class Derived1 : public Base {
    public:
        Derived1(int a);
        virtual void print();
    private:
        int identity;
};

class Derived2 : public Base {
    public:
        Derived2(float x);
        virtual void print();
    private:
        float value;
};

void Base::say_hello(){
    cout << "hello" << endl;
}

Derived1::Derived1(int a){
    identity=a;
}
```
void Derived1::print(){
    cout << "my identity is " << identity << endl;
}

Derived2::Derived2(float x){
    value=x;
}

void Derived2::print(){
    cout << "my value is " << value << endl;
}

class System {
    public:
        System();
        System();
        void go();
    private:
        Base* objects[3];
};

System::System(){
    objects[0] = new Derived1(3);
    objects[1] = new Derived2(10.5);
    objects[2] = new Derived1(5);
}

System::~System(){
    for(int i=0; i<3; i++){
        delete objects[i];
    }
}

void System::go(){
    for(int i=0; i<3; i++){
        objects[i]->say_hello();
        objects[i]->print();
    }
}

int main(){
    System s;
    s.go();
}

Figure 1: C++ program for question 1
Exceptions

2. The program shown in Fig. 2 shows how user exception objects may be defined and used in a program. This program defines a `divide` function, together with an exception to indicate a divide by zero.

```cpp
#include <iostream>
#include <string>
using namespace std;

class DivZero {
public:
    DivZero() { message="Division by Zero!";}
    void print() const { cout << message << endl;}
private:
    string message;
};

int divide(int top, int bottom)
{
    if (bottom == 0)
        throw DivZero();
    return top/bottom;
}

int main(int argc, char* argv[])
{
    try {
        cout << divide (3,2) << endl;
        cout << divide (3,0) << endl;
    } catch(DivZero error) {
        error.print();
    }
}
```

Figure 2: Program showing exceptions

(a) † What will be printed when this program runs?

(b) Define another exception object `NotExact` which will be used to indicate that the numbers do not divide exactly (i.e. the remainder is not zero). Modify `divide` to throw this exception when appropriate, and add another `catch` statement to `main` to handle this exception.

*Hint:* Each `try` can have several `catch` statements after it i.e.

```cpp
   try{
      ...
   } catch(ExA e) {
      ...
   } catch(ExB e) {
      ...
   }
```

(c) * How could you restructure the exception classes so that only one `catch` statement is needed?
3. Fig. 3 shows a class diagram describing part of some software to provide a display at bus stops. Draw a sequence diagram showing what happens when the \texttt{UpdateBusLocation} function is called on \texttt{BusStopDisplay}.

![Class Diagram](image.png)

Figure 3: Bus Stop Display class diagram

Tom Drummond, February 2005
Steve Young, January 2008
Edward Rosten, January 2011