1. (a) The output will be:

```plaintext
hello
my identity is 3
hello
my value is 10.5
hello
my identity is 5
```

Points to emphasise include

- virtual void print()=0;: The = 0 at the end tells us that print is a pure virtual function (i.e., one that has no implementation - it merely defines an interface). The existence of this function in Base ensures that Base is an abstract base class (i.e., no objects of type Base will exist in the program).

- It is possible to include non virtual functions in an abstract base class (e.g., say_hello). The two derived classes inherit this function and can use it. Hence when System runs through the array of three objects both the type-independent say_hello function and the polymorphic print function can be called for each object.

(b) The class diagram is shown below:
(c) The sequence diagram is shown below:

2. (a) The program will output

```
1
Division by Zero!
```

(b) The NotExact exception would be identical to the DivZero exception apart from a change of name and error message. The `divide` function might then be

```cpp
int divide(int top, int bottom)
{
    if (bottom == 0)
        throw DivZero();
    if (top % bottom != 0)
        throw NotExact();
    return top/bottom;
}
```

An extra catch statement would then be required in the main program
int main(int argc, char* argv[]) {
    try {
        cout << divide (3,2) << endl;
        cout << divide (3,0) << endl;
    }
    catch(DivZero error) {
        error.print();
    }
    catch(NotExact error) {
        error.print();
    }
}

(c) The simplest way to avoid multiple catches is to use one exception type parameterised with the type of error eg

    class DivError {
    public:
        DivError(string errm) { message=errm;}
        void print() const { cout << message << endl;}
    private:
        string message;
    }
    ....
    throw DivError("Division by Zero!"); // etc

3. The sequence diagram is shown below:

Tom Drummond, February 2005
Steve Young, January 2008