CISP 401
Introduction to OOP - Java
Dixon

UML - Class Modeling

Object Oriented programming views the solution space design as a hierarchy of cooperating classes. Universal Modelling Language (UML) provides a model for representing classes and class relationships.

A class diagram is one of many models under UML. In a class diagram, the designer expresses the following attributes for each class within the design:

1) class name
2) class attribute
3) class relationship(s)

The class name should be a self documenting name reference and may be expressed graphical(see Introduction to UML pdf) or textually. The class attributes are the data and code members of the class. Each attributes must specify an attribute(member) visibility. Use the table below.

(+) public member (data or code)
(-) private member
(#) protected member

The class relationship(s) describes how each class within the design relates to the other classes using a multiplicity value and a relation. Multiplicities are typically 1:1, 1:m, or m:n and specify the number of object interactions between two classes.

```
ClassAObj --- 1 : m (contains) --- ClassBObj
```

For example, the notation above specifies that an object named ClassAObj contains multiple distinct(1 up to m)) instances of ClassBObj objects.

Typical relations are uses, includes, contains, and inherits.

```
ClassAObj --- 1 : 1 (uses) --- ClassBOjb
ClassAObj --- 1 : m (contains) --- ClassBObj
ClassAObj --- 1 : 6 (contains) --- ClassBObj
ClassAObj --- 1 : 2 (includes) --- ClassBObj
ClassAObj --- 1 : 1 (inherits) ---> ClassBObj
```

uses if a classA sends a message to classB, classA "uses" the services provided by classB.

includes if classA declares an instance of classB as a data member, classA includes classB within its definition.

contains if classA collects and manages multiple instances of classB, classA is a container class for classB.

inherits if classA extends its data and/or services by acquiring classB, classA inherits the data and code of classB.

A class diagram is one of the most useful models for illustrating object oriented design. Using the class diagram, the following represents possible designs for Assign 2a problems.

Assign 2a-1) Class Diagram

Number

```
(-) value
(+) Number() // default constructor
(+) Number(v) // parametized constructor
(+) Number(Number) // copy constructor
(+) datatype get() // accessor;note,datatype is  an appropriate type
(+) void set(v) // manipulator
(+) string toString() // provides external form of Number
```

NumberSystem

```
(-) Number n1, n2, result
(+) NumberSystem()
(+) NumberSystem(Number n1)
(+) NumberSystem(Number n1,Number n2)
(+) NumberSystem(NumberSystem n)
(+) void add()
(+) void subtract()
(+) void multiply()
(+) void divide()
(+) void modulo()
(+) void invert(Number n)
(+) Number getN1()
(+) Number getN2()
(+) Number getResult()
```

Class Associations

```
NumberSystem --- 1 : 3(contains) --- Number
AppDrier  --- 1 : 1(uses) --- AppDriver
```

Assign 2a-2) Class Diagram

ComplexNumber

```
(-) realPart, imaginaryPart
(+) ComplexNumber() // default constructor
(+) ComplexNumber(rPart,Ipart) // parametized constructor
(+) ComplexNumber(ComplexNumber) // copy constructor
(+) float getReal() // accessors
(+) float getImag()
(+) void setReal(rp) // manipulators
(+) void setImag(ip)
(+) string toString() // provides external form of ComplexNumber
```

ComplexNumberSystem

```
(-) ComplexNumber n1, n2, result
(+) ComplexNumberSystem()
(+) ComplexNumberSystem(ComplexNumber n1)
(+) ComplexNumberSystem(ComplexNumber n1,ComplexNumber n2)
(+) ComplexNumberSystem(ComplexNumberSystem cns)
(+) void add()
(+) void subtract()
(+) void multiply()
(+) void divide()
(+) ComplexNumber getN1()
(+) ComplexNumber getN2()
(+) ComplexNumber getResult()
```

Class Associations

```
ComplexNumberSystem --- 1 : 3(contains) --- ComplexNumber
AppDriver --- 1 : 1(uses) --- ComplexNumberSystem
```

Assign 2a-3) Class Diagram

MyStringClass

```
(-) byte[] privateString;
(+) MyStringClass()
(+) MyStringClass(string s)
(+) MyStringClass(MyStringClass msC)
(+) MyStringClass left (short size)
(+) MyStringClass right(short size)
(+) MyStringClass mid(short startIndex, short size)
(+) short instr(short startIndex, MyStringClass searchString)
(+) MyStringClass upper()
(+) MyStringClass lower()
(+) MyStringClass invert()
(+) string toString()
```

Class Associations

AppDriver --- 1 : 1(uses) --- MyStringClass