University of Technology, Jamaica
School of Computing and Information Technology

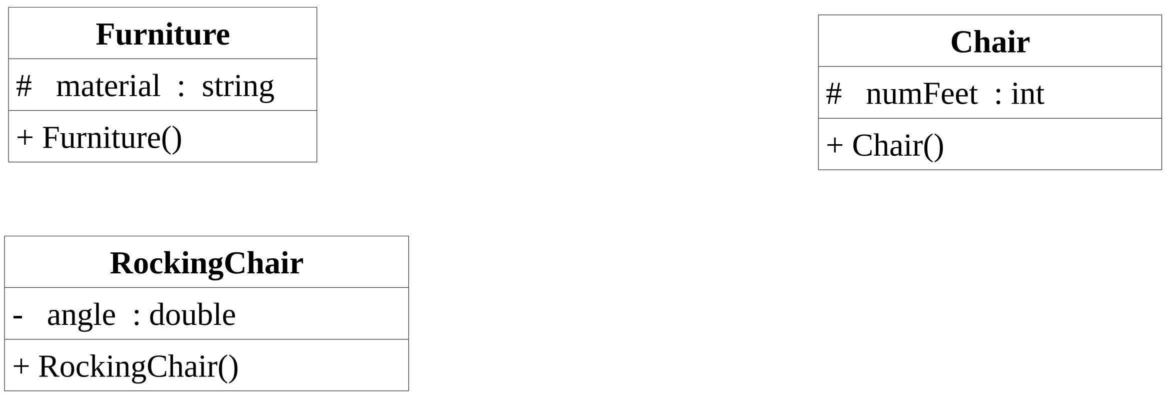
OOP Principles Lab Practical Six - Inheritance Part Two

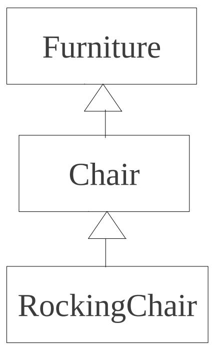
Objective:
The objectives of this lab practical are to allow students to be able to:

- Expand their experience working with inheritance hierarchies
- Explore the order in which constructors are called in inheritance hierarchies
- Calling parent class constructors and other method from child class objects
- Working with concrete, abstract and overridable methods
- Implementing multiple inheritance in C++ and interfaces in Java

This lab demonstrates aims to build the competence of students in implementing the powerful inheritance features of the OOP paradigm, making use of code reuse. This lab will pave the way for the next lecture, tutorial and lab classes on polymorphism.

Design using UML





Exercise One

Enter the C++ and Java programs below which implement an inheritance hierarchy (similar to the one shown in lecture six – inheritance part two). Each class and the driver are to be placed in separate files. The driver instantiates an object for each of the three classes. Before you run the program, try to figure out which constructors will be called and what order they will be called in. Now run the program and compare the printed results with what you had expected.

C++

```
//Furniture.h
#ifndef FurnitureID
#define FurnitureID
#include <string>
#include <iostream>
using namespace std;

//Furniture class
class Furniture
{
      protected:
            string material;

      public:
            Furniture()
            {
                  material = "No material specified";
                  cout << "Printing from within the Furniture constructor" << endl;
            }
};

#endif

//Chair.h
#ifndef ChairId
#define ChairId
#include "Furniture.h"
#include <iostream>
using namespace std;
//Chair class
class Chair : public Furniture
{
      protected:
            int numFeet;

      public:
            Chair()
            {
                  material = "wood";
                  numFeet  = 4;
                  cout << "Printing from within the Chair constructor" << endl;
            }
};
#endif
```

```
//RockingChair.h
#ifndef RockingChairId
#define RockingChairId
#include "Chair.h"
#include <iostream>
using namespace std;
//RockingChair class
class RockingChair : Chair
{
      private:
            double tiltAngle;

      public:
            RockingChair()
            {
                  material = "wood";
                  numFeet  = 4;
                  tiltAngle = 30.0;
                  cout << "Printing from within the RockingChair constructor" <<
endl;
            }
};
#endif

//Driver.java
public class Driver
{
      public static void main(String[] args)
      {
            System.out.println("\nCreating a Furniture object");
            Furniture fObj = new Furniture();
            System.out.println("\nCreating a Chair object");
            Chair cObj = new Chair();
            System.out.println("\nCreating a RockingChair object");
            RockingChair rObj = new RockingChair();
      }
}
```

Java

```
//Furniture.java
//Furniture class
public class Furniture
{
      protected String material;

      public Furniture()
      {
            material = "No material specified";
            System.out.println("Printing from within the Furniture constructor");
      }
}
//Chair.java
//Chair class
public class Chair extends Furniture
{
      protected int numFeet;

      public Chair()
      {
            material = "wood";
            numFeet  = 4;
            System.out.println("Printing from within the Chair constructor");
      }
}
//RockingChair.java
//RockingChair class
public class RockingChair extends Chair
{
      private double tiltAngle;

      public RockingChair()
      {
            material = "wood";
            numFeet  = 4;
            tiltAngle = 30.0;
            System.out.println(
                  "Printing from within the RockingChair constructor");
      }
}
//Driver.java
public class Driver
{
      public static void main(String[] args)
      {
            System.out.println("\nCreating a Furniture object");
            Furniture fObj = new Furniture();
            System.out.println("\nCreating a Chair object");
            Chair cObj = new Chair();
            System.out.println("\nCreating a RockingChair object");
            RockingChair rObj = new RockingChair();
      }
}
```

Exer Two

Add an abstract method called Info( ) to the Furniture class. It should take no arguments and return no value. What are the implications of adding this abstract method to the base class? Do you expect the program to compile now? Why or why not?

C++

```cpp
//Furniture.h
#ifndef FurnitureID
#define FurnitureID
#include <string>
#include <iostream>
using namespace std;
//Furniture class
class Furniture
{
	protected:
		string material;

	public:
		Furniture()
		{
			material = "No material specified";
			cout << "Printing from within the Furniture constructor" << endl;
		}
		virtual void Info() = 0; //abstract method
};

#endif
```

Java

```java
//Furniture.java
//Furniture class
public abstract class Furniture
{
	protected String material;

	public Furniture()
	{
		material = "No material specified";
		System.out.println("Printing from within the Furniture constructor");
	}
	public abstract void Info(); //abstract method
}
```

Comment

Adding an abstract class to the base class means that all derived classes will become abstract unless they implement the method in their own classes. So the Furniture class will become abstract because of the abstract method, and the Chair class will also become abstract because it inherits from the parent class Furniture and does not implement Info(). The RockingChair class will also be abstract because it too does not provide an implementation for Info(). The program will not compile because you can instantiate objects from abstract classes and now all three classes have become abstract.

Exercise Three

Modify your program so the Chair and RockingChair classes are no longer abstract, by providing an implementation for Info() in both. Modify main() in the driver so that the program can now be run. Do not delete any lines – instead comment any line that is now illegal.

C++

```
//Chair.h
#ifndef ChairId
#define ChairId
#include "Furniture.h"
#include <iostream>
using namespace std;
//Chair class
class Chair : public Furniture
{
      protected:
            int numFeet;

      public:
            Chair()
            {
                  material = "wood";
                  numFeet  = 4;
                  cout << "Printing from within the Chair constructor" << endl;
            }
            void Info()
            {
                  cout << "This chair is made of " << material <<
                              " and has " << numFeet << " number of feet."<< endl;
            }
};
#endif

//RockingChair.h
#ifndef RockingChairId
#define RockingChairId
#include "Chair.h"
#include <iostream>
using namespace std;
//RockingChair class
class RockingChair : Chair
{
      private:
            double tiltAngle;

      public:
            RockingChair()
            {
                  material = "wood";
                  numFeet  = 4;
                  tiltAngle = 30.0;
                  cout << "Printing from within the RockingChair constructor"
                   << endl;
            }
            void Info()
            {
                  cout << "This chair is made of " << material <<
                              " and has " << numFeet << " number of feet." << endl;
                  cout << "This rocking chair has a tilt angle of "
                         << tiltAngle << endl;

            }
};
#endif

//Driver.cpp
#include <iostream>
using namespace std;
#include "RockingChair.h"
int main ()
{
      //cout << "\nCreating a Furniture object" << endl;
      //Furniture *fObj = new Furniture;
      cout << "\nCreating a Chair object" << endl;
      Chair *cObj = new Chair;
      cObj->Info();
      cout << "\nCreating a RockingChair object" << endl;
      RockingChair *rObj = new RockingChair;
      rObj->Info();
      return 0;
}
```

Java

```
//Chair.java
//Chair class
public class Chair extends Furniture
{
      protected int numFeet;

      public Chair()
      {
            material = "wood";
            numFeet  = 4;
            System.out.println("Printing from within the Chair constructor");
      }
      public void Info()
      {
            System.out.println("This chair is made of "+material+
                        " and has "+numFeet+" number of feet.");
      }
}

//RockingChair.java
//RockingChair class
public class RockingChair extends Chair
{
      private double tiltAngle;

      public RockingChair()
      {
            material = "wood";
            numFeet  = 4;
            tiltAngle = 30.0;
            System.out.println(
                  "Printing from within the RockingChair constructor");
      }
      public void Info()
      {
            System.out.println("This chair is made of "+material+
                        " and has "+numFeet+" number of feet.");
            System.out.println(
                  "This rocking chair has a tilt angle of "+tiltAngle);
      }
}

//Driver.java
public class Driver
{
      public static void main(String[] args)
      {
            //System.out.println("\nCreating a Furniture object");
            //Furniture fObj = new Furniture();
            System.out.println("\nCreating a Chair object");
            Chair cObj = new Chair();
            cObj.Info();
            System.out.println("\nCreating a RockingChair object");
            RockingChair rObj = new RockingChair();
            rObj.Info();
        }
}
```

Exercise Four

Notice that in the overridden version of Info() in the RockingChair class, the code to print “`This chair is made of wood and has 4 number of feet.`” is repeated – it is the same code that is in the Info() method in the Chair class. However, one of the key benefits of inheritance in OOP programs is to allow reuse of code. Take advantage of the inheritance relationship between the Chair and RockingChair classes by replacing the repeated line in the RockingChair class with a call to the parent (Chair) class' Info() method.

C++

```
//RockingChair.h
#ifndef RockingChairId
#define RockingChairId
#include "Chair.h"
#include <iostream>
using namespace std;

//RockingChair class
class RockingChair : Chair
{
      private:
            double tiltAngle;

      public:
            RockingChair()
            {
                  material = "wood";
                  numFeet  = 4;
                  tiltAngle = 30.0;
                  cout << "Printing from within the RockingChair constructor"
                        << endl;
            }
            void Info()
            {
                  Chair::Info();
                  cout << "This rocking chair has a tilt angle of " << tiltAngle
                         << endl;

            }
};

#endif
```

Java

```java
//RockingChair.java
//RockingChair class
public class RockingChair extends Chair
{
      private double tiltAngle;

      public RockingChair()
      {
            material = "wood";
            numFeet  = 4;
            tiltAngle = 30.0;
            System.out.println(
                  "Printing from within the RockingChair constructor");
      }
      public void Info()
      {
            super.Info();
            System.out.println(
                  "This rocking chair has a tilt angle of "+tiltAngle);
      }
}
```

Homework

a) Add a primary constructor to each of the three classes.

b) Change the access specifier for the material attribute in the Furniture class from protected to private. Recompile the program. Why wont the program compile? Instead of changing the access specifier back to protected, force a call from the child class constructor to the parent class constructor and recompile. Your program should run now.

c) Create an Item class in C++ and an Item interface in Java. Item should have one abstract method called GetDescription( ). Using multiple inheritance in C++ and inheritance plus interface in Java, allow the RockingChair class to inherit/implement the Item class/interface. (See lecture notes six for help).