ECE 462 Exam 1

6:30-7:30PM, September 22, 2010

I will not receive nor provide aid to any other student for this exam.

Signature:

You must sign here. Otherwise, the exam is not graded.

This exam is printed **double sides**.

Write your answers next to the questions. If you need more space, you can use the two blank pages.

This is an *open-book, open-note* exam. You can use any book or note or program printouts.

Please turn off your cellular phone **now** .

Two outcomes are tested in this exam. To pass each outcome, you must receive 50% of the points.

- Outcome 3: an understanding of the concepts of inheritance and polymorphism.
- Outcome 4: an ability to use template classes and libraries in C++ and Java.

You need to obtain 50% of the points in each outcome to pass the outcome. If a program has a syntax error, **point out** which line causes the error.

If a question has a numeric answer, you can write the *procedure* without the final result. For example, you can write "1 + 2" instead of "3".

Contents

Passed Outcomes: 3 4

Total Score:

This page is blank.

This page is blank.

1 Abstract Class (1 point)

Which statement is correct? You may choose multiple answers.

Answer: A

A. If a Java class has an abstract member function (i.e. method), this class is abstract.

B. If a Java class is abstract, **all** methods of this class must be abstract.

C. If a C++ class has a pure virtual function, **all** methods must be virtual.

D. If a C++ class has a pure virtual function, **all** methods must be pure virtual.

E. If a Java class has an abstract method, this class must not have any private attribute. All attributes must be protected.

2 C++ Inheritance and Polymorphism (Outcome 3, 4 points)

Answer the questions based on the following class hierarchy. There are 16 answers, 0.25 point for each answer.

```
// question31.h
#ifndef QUESTION3_H
#define QUESTION3_H
#include <iostream>
using namespace std;
class Base
{
public:
  void f1() { cout << "B:f1 "; f2(); } // CAUTION: call f2
  void f2() { cout << "B:f2"<< endl; }
  virtual void f3() { cout << "B:f3"<< endl; }
};

class Derived1: public Base
{
public:
  void f1() { cout << "D1:f1 "; f2(); } // CAUTION:  all f2
  virtual void f2() { cout << "D1:f2"<< endl; }
  virtual void f4() { cout << "D1:f4"<< endl; }
};

class Derived2: public Derived1
{
public:
  void f2() { cout << "D2:f2"<< endl; }
  void f3() { cout << "D2:f3"<< endl; }
  void f5() { cout << "D2:f5 ";  f2(); } // CAUTION: call f2
};

#endif
// end of question31.h

//
// question31.cpp
//
#include "question31.h"
int main(int argc, char * argv[])
{
  // Which lines cause compile-time errors?
  Base * b2d1 = new Derived1;
  Base * b3d2 = new Derived2;
```

```
  Derived1 * d1b1 = new Base;
  Derived1 * d1d1 = new Derived1;
  Derived1 * d1d2 = new Derived2;
  Derived2 * d2d2 = new Derived2;

  // Which lines below cause compile-time errors?
  // If a line has no error, write the outputs.
  // Please write your answers next to the corresponding lines.

  // If a variable has an error earlier, ignore the variable below.
  // For example, if d1b1 has an error, ignore everything about d1b1.

  b2d1 -> f3();

  b3d2 -> f2();

  b3d2 -> f1();

  b3d2 -> f5();

  d1d1 -> f1();

  d1d2 -> f4();

  d2d2 -> f1();

  d2d2 -> f2();

  d2d2 -> f4();

  d2d2 -> f5();

  return 0;
}
```

Answer:

```
errors:
Derived1 * d1b1 = new Base;
b3d2 -> f5();


b2d1 -> f3(); B:f3
b3d2 -> f2(); B:f2
b3d2 -> f1(); B:f1 B:f2
b3d2 -> f5(); error
```

```
d1d1 -> f1(); D1:f1 D1:f2
d1d2 -> f4(); D1:f4
d2d2 -> f1(); D1:f1 D2:f2
d2d2 -> f2(); D2:f2
d2d2 -> f4(); D1:f4
d2d2 -> f5(); D2:f5 D2:f2
```

3 Java Inheritance and Polymorphism (Outcome 3, 2 points)

Answer the questions based on the following class hierarchy. There are 5 answers, 0.4 point for each answer.

```
class Base
{
    public void f1() { System.out.print("B:f1 "); f2(); }
    public void f2() { System.out.println("B:f2");  }
    public void f3() { System.out.println("B:f3");  }
}

class Derived1 extends Base
{
    public void f1() { System.out.print("D1:f1 "); f2(); }
    public void f2() { System.out.println("D1:f2");  }
    public void f4() { System.out.println("D1:f4");  }
}

class Derived2 extends Derived1
{
    public void f2() { System.out.println("D2:f2");  }
    public void f3() { System.out.println("D2:f3");  }
    public void f5() { System.out.print("D2:f5 "); f2(); }
}

class Question
{
    public static void main(String [] args)
    {
        Base b2d1 = new Derived1();
        Base b3d2 = new Derived2();
        Derived1 d1d1 = new Derived1();
        Derived1 d1d2 = new Derived2();
        Derived2 d2d2 = new Derived2();

        // Write the outputs next to the corresponding lines.

        b3d2.f2();

        b3d2.f1();

        d1d1.f1();

        d2d2.f1();
```

```
        d2d2.f5();

    }
}
```

Answer:

```
no compile-time error

b3d2.f2(); D2:f2
b3d2.f1(); D1:f1 D2:f2
d1d1.f1(); D1:f1 D1:f2
d2d2.f1(); D1:f1 D2:f2
d2d2.f5(); D2:f5 D2:f2
```

4 C++ Template (Outcome 4, 2 point)

Use arrays to implement a map. A map stores (key, value) pairs. The keys must be distinct. This map **can handle different types**, including `int` and `string`; therefore, you should use `template`.

```
#include <iostream>
#include <string>
using namespace std;
// ---> FIX ME <---
// create generic map using template
// a map stores (key, value) pairs
class ECE462MAP
{
public:
  ECE462MAP()
  {
    size = 10;
    item = 0;
    kArray = new KEY[size];
    vArray = new VALUE[size];
  }
  bool contains(KEY k)
  {
    // ---> FIX ME <---
    // does the key already exist?
    // return true, if it does
    // return false, if it does not
  }
  void insert(KEY k, VALUE v)
  {
    // check whether the key has already been seen
    for (int i = 0; i < item; i ++)
      {
        if (kArray[i] == k)
          {
            // replace the value
            vArray[i] = v;
            return;
          }
      }
    // new key
    if (item == size)
      {
        // need more space
        size *= 2;
```

```
      KEY * kArray2 = new KEY[size];
      VALUE * vArray2 = new VALUE[size];
      // copy elements
      for (int i = 0; i < item; i ++)
        {
          kArray2[i] = kArray[i];
          vArray2[i] = vArray[i];
        }
      delete [] kArray;
      delete [] vArray;
      kArray = kArray2;
      vArray = vArray2;
    }
  kArray[item] = k;
  vArray[item] = v;
  item ++;
}
// ---> FIX ME <---
// write the return type and the argument type of the "get" function
{
  // assume the map contains this key
  for (int i = 0; i < item; i ++)
    {
      if (kArray[i] == k)
        {
          return vArray[i];
        }
    }
  // should not reach here
  return vArray[0];
}
void remove(KEY k)
{
  // assume the set contains this key
  for (int i = 0; i < item; i ++)
    {
      if (kArray[i] == k)
        {
          kArray[i] = kArray[item - 1];
          vArray[i] = vArray[item - 1];
          item --;
          return;
        }
    }
  // don't worry about shrinking the arrays
```

```
    // should not reach here
  }
  int getSize()
  {
    return item;
  }
  virtual ~ECE462MAP()
  {
    // ---> FIX ME <---
    // release memory for the key and value arrays
  }

private:
  KEY * kArray;
  VALUE * vArray;
  int size;
  int item;
};
int main(int argc, char * argv[])
{
  // create a map string -> integer
  ECE462MAP<string,int> mapsi;
  mapsi.insert("ece", 4);
  mapsi.insert("purdue", 6);
  mapsi.insert("lafayette", 2);
  cout << mapsi.get("lafayette") << endl; // 2
  mapsi.insert("lafayette", 9);
  cout << mapsi.get("lafayette") << endl; // 9
  cout << mapsi.getSize() << endl;        // 3
  mapsi.remove("purdue");
  cout << mapsi.getSize() << endl;        // 2
  cout << mapsi.contains("ece") << endl;  // 1
  // create a map integer -> string
  ECE462MAP<int, string> mapis;
  mapis.insert(2, "WEST");
  cout << mapis.getSize() << endl;        // 1
  cout << mapis.get(2) << endl;           // WEST
  // create a map string -> string
  ECE462MAP<string,string> mapss;
  mapss.insert("Hello", "Purdue");
  cout << mapss.get("Hello") << endl;     // Purdue
  return 0;
}
```

Answer:

```
#include <iostream>
#include <string>
using namespace std;
template<class KEY, class VALUE> class ECE462MAP
{
public:
  ECE462MAP()
  {
    size = 10;
    item = 0;
    kArray = new KEY[size];
    vArray = new VALUE[size];
  }
  bool contains(KEY k)
  {
    for (int i = 0; i < item; i ++)
      {
        if (kArray[i] == k)
          {
            return true;
          }
      }
    return false;
  }
  void insert(KEY k, VALUE v)
  {
    for (int i = 0; i < item; i ++)
      {
        if (kArray[i] == k)
          {
            vArray[i] = v;
            return;
          }
      }
    if (item == size)
      {
        size *= 2;
        KEY * kArray2 = new KEY[size];
        VALUE * vArray2 = new VALUE[size];
        for (int i = 0; i < item; i ++)
          {
            kArray2[i] = kArray[i];
            vArray2[i] = vArray[i];
          }
        delete [] kArray;
```

```
        delete [] vArray;
        kArray = kArray2;
        vArray = vArray2;
      }
    kArray[item] = k;
    vArray[item] = v;
    item ++;
  }
  VALUE get(KEY k)
  {
    for (int i = 0; i < item; i ++)
      {
        if (kArray[i] == k)
          {
            return vArray[i];
          }
      }
    return vArray[0];
  }
  void remove(KEY k)
  {
    for (int i = 0; i < item; i ++)
      {
        if (kArray[i] == k)
          {
            kArray[i] = kArray[item - 1];
            vArray[i] = vArray[item - 1];
            item --;
            return;
          }
      }
  }
  int getSize()
  {
    return item;
  }
  virtual ~ECE462MAP()
  {
    delete [] kArray;
    delete [] vArray;
  }

private:
  KEY * kArray;
  VALUE * vArray;
```

```
  int size;
  int item;
};
int main(int argc, char * argv[])
{
  // create a map string -> integer
  ECE462MAP<string,int> mapsi;
  mapsi.insert("ece", 4);
  mapsi.insert("purdue", 6);
  mapsi.insert("lafayette", 2);
  cout << mapsi.get("lafayette") << endl; // 2
  mapsi.insert("lafayette", 9);
  cout << mapsi.get("lafayette") << endl; // 9
  cout << mapsi.getSize() << endl;        // 3
  mapsi.remove("purdue");
  cout << mapsi.getSize() << endl;        // 2
  cout << mapsi.contains("ece") << endl;  // 1
  // create a map integer -> string
  ECE462MAP<int, string> mapis;
  mapis.insert(2, "WEST");
  cout << mapis.getSize() << endl;        // 1
  cout << mapis.get(2) << endl;           // WEST
  // create a map string -> string
  ECE462MAP<string,string> mapss;
  mapss.insert("Hello", "Purdue");
  cout << mapss.get("Hello") << endl;     // Purdue
  return 0;
}
```

5 C++ Iterator (Outcome 4, 2 point)

Write the `print` function using an iterator.

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

void print(vector<int> vec)
{
  /* ---> FIX ME <--- */
  /* iterate through the elements and print the elements one by one */
  cout << endl;
}


int main()
{
  int data[] = {11, 12, 23, 34};
  int size = sizeof( data ) / sizeof( data[0] );
  vector<int> vec;
  for (int i = 0; i < size; i ++)
    {
      vec.push_back(data[i]);
    }
  print( vec );
  return 0;
}
```

Answer:

```
void print(vector<int> vec )
{
  vector<int>::iterator p = vec.begin();
  while ( p < vec.end() )
    {
      cout <<  *p++ << " ";
    }
  cout << endl;
}
```

6 Java List and Class Hierarchy (Outcome 4, 2 point)

What is the output of this program? There are four answers, 0.5 point for each.

```
import java.util.*;
class Base
{
    public void f1() { System.out.print("B:f1 "); f2(); }
    public void f2() { System.out.println("B:f2");  }
}

class Derived1 extends Base
{
    public void f1() { System.out.print("D1:f1 "); f2(); }
}

class Derived2 extends Derived1
{
    public void f2() { System.out.println("D2:f2");  }
}

class Question
{
    public static void main(String [] args)
    {
        List<Base> blist = new ArrayList<Base>();
        Base b1b = new Base();
        Base b2d1 = new Derived1();
        Base b3d2 = new Derived2();
        Derived1 d1d2 = new Derived2();
        blist.add(b1b);
        blist.add(b2d1);
        blist.add(b3d2);
        blist.add(d1d2);
        ListIterator iter = blist.listIterator();
        while ( iter.hasNext() )
            {
                Base b = (Base) iter.next();
                b.f1();
            }
    }
}
```

Answer:

```
B:f1 B:f2
```

```
D1:f1 B:f2
D1:f1 D2:f2
D1:f1 D2:f2
```

7 C++ Constructor and Destructor (2 point)

Write down the output of this program. Make sure you draw the separators (-----).

Answer:

```
B1::B1
D11::D11
-----
B2::B2
D21::D21
D22::D22
-----
B1::~B1
-----
D22::~D22
D21::~D21
B2::~B2
```

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

class B1
{
public:
  B1() { cout << "B1::B1" << endl; }
  ~ B1() { cout << "B1::~B1" << endl; }
};

class D11: public B1
{
public:
  D11() { cout << "D11::D11" << endl; }
  ~ D11() { cout << "D11::~D11" << endl; }
};

class D12: public D11
{
public:
  D12()
  {
    cout << "D12::D12" << endl;
    data = new int[100];
  }
  ~ D12()
  {
```

```
    cout << "D12::~D12" << endl;
    delete [] data;
  }
private:
  int * data;
};

class B2
{
public:
  B2() { cout << "B2::B2" << endl; }
  virtual ~ B2() { cout << "B2::~B2" << endl; }
};

class D21: public B2
{
public:
  D21()
  {
    cout << "D21::D21" << endl;
    data = new int[100];
  }
  ~ D21()
  {
    cout << "D21::~D21" << endl;
    delete [] data;
  }
  int * data;
};

class D22: public D21
{
public:
  D22() { cout << "D22::D22" << endl; }
  ~ D22() { cout << "D22::~D22" << endl; }
};

int main(int argc, char * argv[])
{
  B1 * b1 = new D11;
  cout << "-----" << endl;
  B2 * b2 = new D22;
  cout << "-----" << endl;
  delete b1;
  cout << "-----" << endl;
```

```
  delete b2;
  return 0;
}
```