Experiment 8
Polymorphism

Objectives

- How polymorphism makes programming more convenient and systems more extensible.
- The distinction between abstract and concrete classes and how to create abstract classes.
- To use runtime type information (RTTI).
- How C++ implements virtual functions and dynamic binding.
- How virtual destructors ensure that all appropriate destructors run on an object.
- How polymorphism makes programming more convenient and systems more extensible.
- The distinction between abstract and concrete classes and how to create abstract classes.

Prelab Activities

Programming Output

For each of the given program segments, read the code and write the output in the space provided below each program. [Note: Do not execute these programs on a computer.]

For Programming Output Exercises 1–2, use the following class definition.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  // class Oyster definition
6  class Oyster
7  {
8  public:
9     // constructor
10    Oyster( string genusString )
11    {
12       genus = genusString;
13    } // end class Oyster constructor
14
15    // function getPhylum definition
16    string getPhylum() const
17    {
18       return "Mollusca";
19    } // end function getPhylum
20
21    // function getName definition
22    virtual string getName() const
23    {
24       return "Oyster class";
25    } // end function getName
26
27    // function getGenus definition
28    string getGenus() const
29    {
30       return genus;
31    } // end function getGenus
32
```

```
33    // print function
34    virtual void print() const = 0;
35 private:
36    string genus;
37 }; // end class Oyster
38
39 // class VirginiaOyster definition
40 class VirginiaOyster : public Oyster
41 {
42 public:
43    // constructor calls base-class constructor
44    VirginiaOyster()
45       : Oyster( "Crassostrea" )
46    {
47       // empty
48    } // end class VirginiaOyster constructor
49
50    // function getName definition
51    virtual string getName() const
52    {
53       return "VirginiaOyster class";
54    } // end function getName
55
56    // print function
57    virtual void print() const
58    {
59       cout << "Phylum: " << getPhylum()
60            << "\tGenus: " << getGenus();
61    } // end print function
62 }; // end class VirginiaOyster
```

1. What is output by the following program? Use class Oyster and VirginiaOyster.

```
1 #include <iostream>
2 using namespace std;
3
4 #include "Oyster.cpp"
5
6 int main()
7 {
8    VirginiaOyster oyster;
9    Oyster *baseClassPtr;
10
11   baseClassPtr = &oyster;
12   baseClassPtr->print();
13
14   cout << endl;
15 } // end main
```

2. What is output by the following program segment? Assume that the Oyster class member function print has been changed to that shown below.

```
1 // function print definition
2 virtual void print() const
3 {
4    cout << "Oysters belong to Phylum " << getPhylum() << endl;
5 } // end function print
```

```
1 #include <iostream>
2 using namespace std;
3
4 #include "oyster.cpp"
5
6 int main()
7 {
8    VirginiaOyster *ptr;
9    VirginiaOyster oyster;
10   Oyster *oysterPtr;
11
12   oysterPtr = &oyster;
13   ptr = &oyster;
14
15   ptr -> print();
16   cout << endl;
17
18   oysterPtr -> print();
19   cout << endl << oysterPtr -> getPhylum();
20
21   cout << endl;
22 } // end main
```

CorrecttheCode

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic or compilation error, write the corrected code. If the code does not contain an error, write "no error." For code segments, assume the code appears in main and that using directives are provided. [Note: It is possible that a program segment may contain multiple errors.]

1. The following code defines an abstract class named Base:

```
1 // class Base definition
2 class Base
3 {
4 public:
5    void print() const;
6 }; // end class Base
```

2. The following is a modified version of the definition of class VirginiaOyster. Assume member function print is defined in another file.

```
1 // class VirginiaOyster definition
2 class VirginiaOyster : public Oyster
3 {
4 public:
5    // constructor
6    virtual VirginiaOyster( string genusString )
7    {
8       genus = genusString;
9    } // end class VirginiaOyster constructor
10
11   // constructor
12   VirginiaOyster( char *genusString )
13   {
14      genus = genusString;
15   } // end class VirginiaOyster constructor
16
17   // print function
18   void print() const;
19 }; // end class VirginiaOyster
```

3. The following program defines two classes—BaseClass and DerivedClass—and instantiates an object of type BaseClass. [Note: Only the definitions for BaseClass and DerivedClass are shown; assume that another file is provided that contains the classes' implementations.]

```
1 // class BaseClass definition
2 class BaseClass
3 {
4 public:
```

```
5     BaseClass( int = 0, int = 0 );
6     virtual void display() = 0;
7  private:
8     int x;
9     int y;
10 }; // end class BaseClass
11
12 // class DerivedClass definition
13 class DerivedClass
14 {
15 public:
16    DerivedClass( int = 0, int = 0, int = 0 );
17    virtual void display();
18 private:
19    int z;
20 }; // end class BaseClass
21
22 int main()
23 {
24    BaseClass b( 5, 10 );
25    b.display();
26 } // end main
```

4. The following program segments define two classes: Name and NameAndWeight. Name should be an abstract base class and NameAndWeight should be a concrete derived class. Function main should declare an object of type NameAndWeight and print its name and weight. [Note: Only the definitions for Name and NameAndWeight are shown; assume files containing member function definitions have been provided elsewhere.]

```
1  // class Name definition
2  class Name
3  {
4  public:
5     Name( string );
6     virtual void printName() const = 0;
7  private:
8     string name;
9  }; // end class Name
10
11 // class NameAndWeight definition
12 class NameAndWeight : public Name
13 {
14 public:
15    NameAndWeight( string, int = 0 );
16    virtual void displayWeight() const;
17 private:
18    int weight;
19 }; // end class NameAndWeight
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6     NameAndWeight object( "name", 100 );
7
8     cout << "name: " << object.printName() << endl;
9     cout << "weight: " << object.displayWeight() << endl;
10 } // end main
```

Lab Exercises

Lab Exercises 1 – Polymorphic Media Players

The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. UML Diagrams
4. Sample Output
5. Test Code
6. Problem-Solving Tips

The test program represents a complete working C++ test program. Read the problem description and examine the sample output; then study the test code. Using the problem-solving tips as a guide write your C++ code. Compile and execute the program. Compare your output with the sample output provided.

Lab Objectives

In this lab, you will practice:

- Creating an Player base class that contains virtual and pure virtual functions and derived classes MusicPlayer, VideoPlayer,Walkman, IPod, BugiVideoPlayer, VLCVideoPlayer.
- Defining virtual functions.
- Calling virtual functions.

Description of the Problem

Develop a polymorphic media players hierarch. Create a Player class as a base class of the system. Inherit other to classes MusicPlayer and VideoPlayer from the Player class. Look at the UML Diagram and Sample Output for hierarchy details.

UML Diagrams

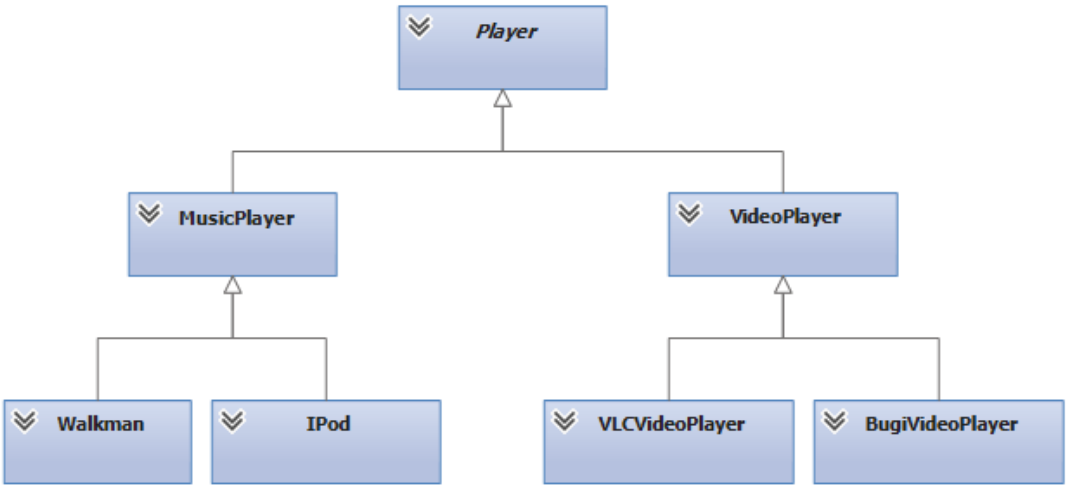

Figure 1: General Diagram

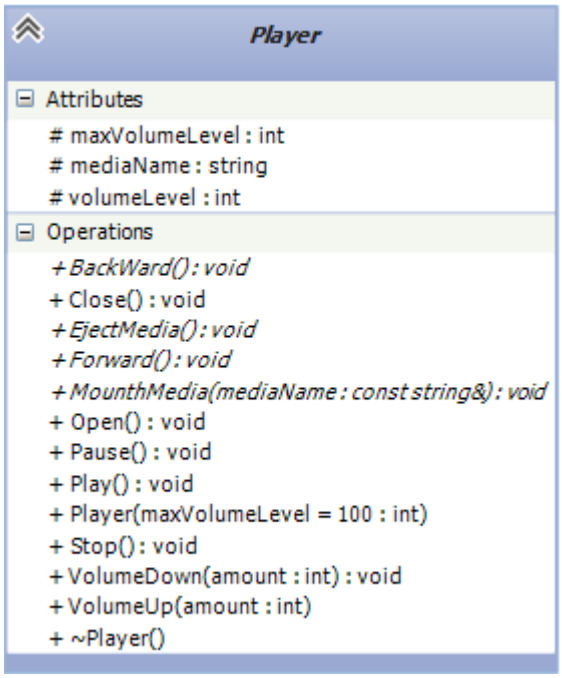

Figure 2:Player class

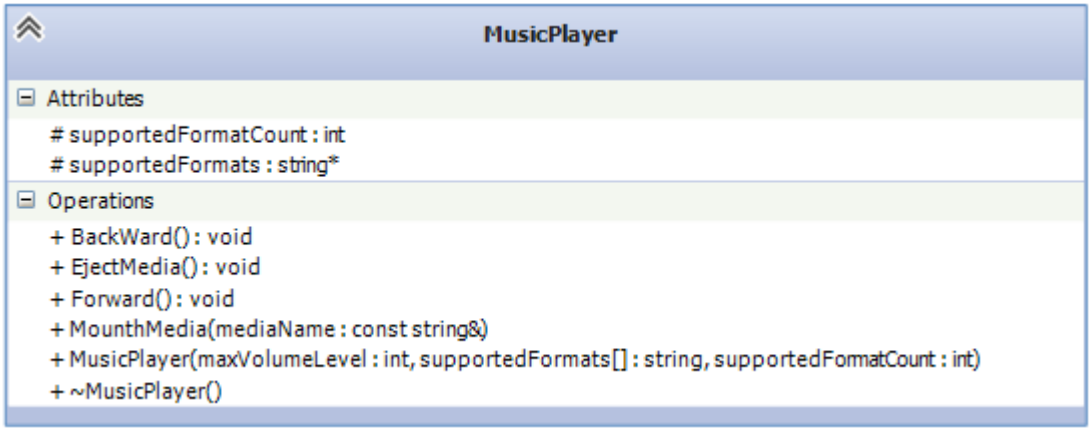

Figure 3:MusicPlayer class

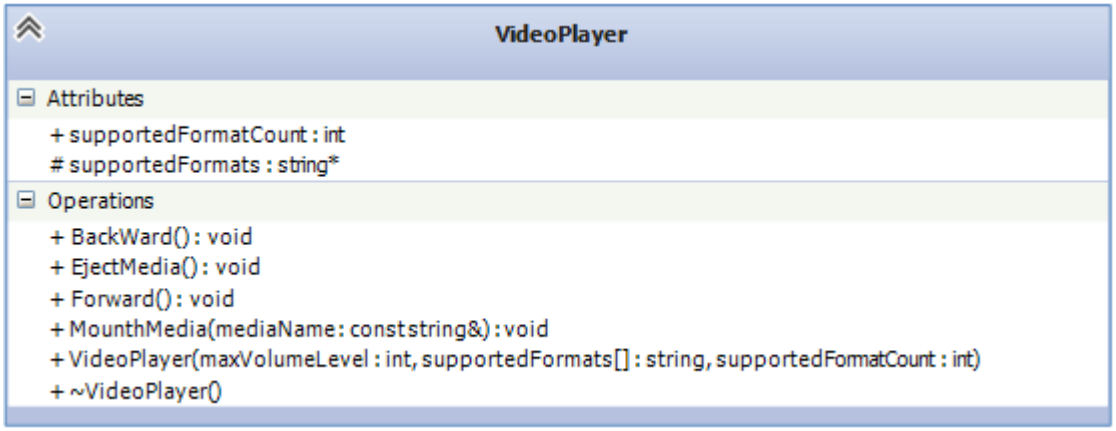

Figure 4:VideoPlayer class

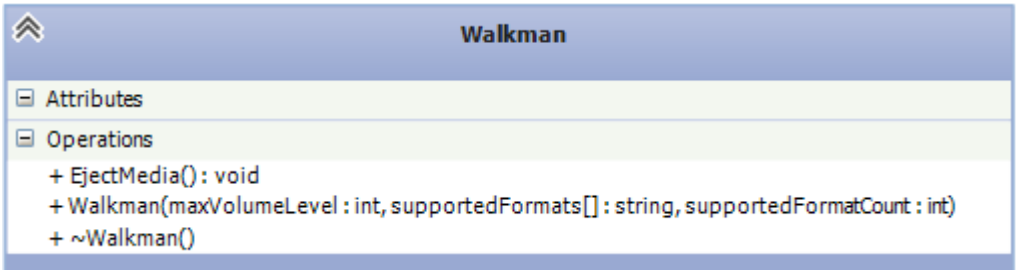

Figure 4:Walkman class

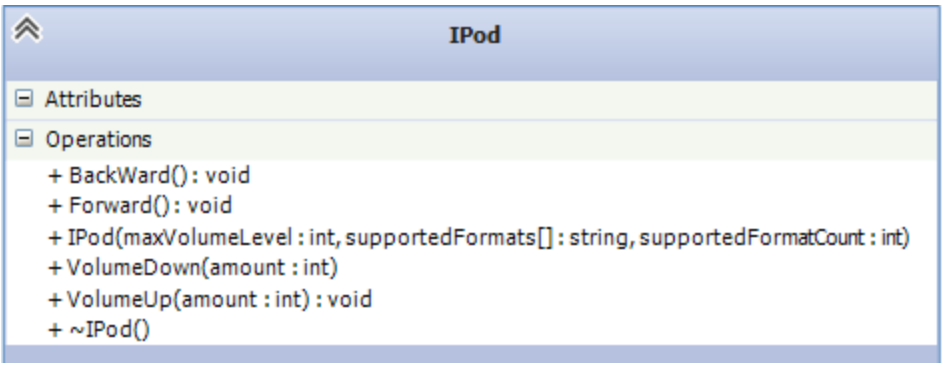

Figure 5:IPod class

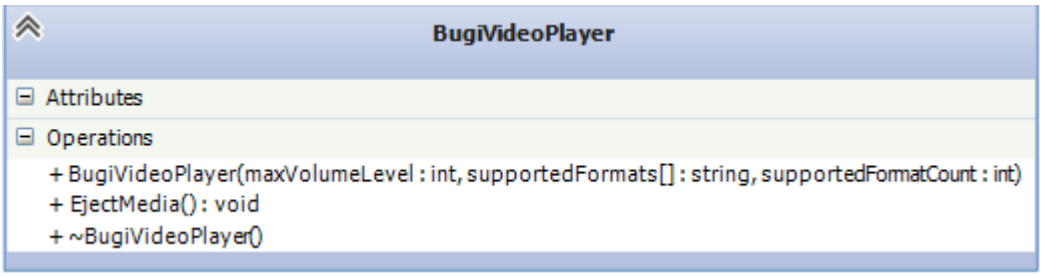

Figure 6:BugiVideoPlayer class

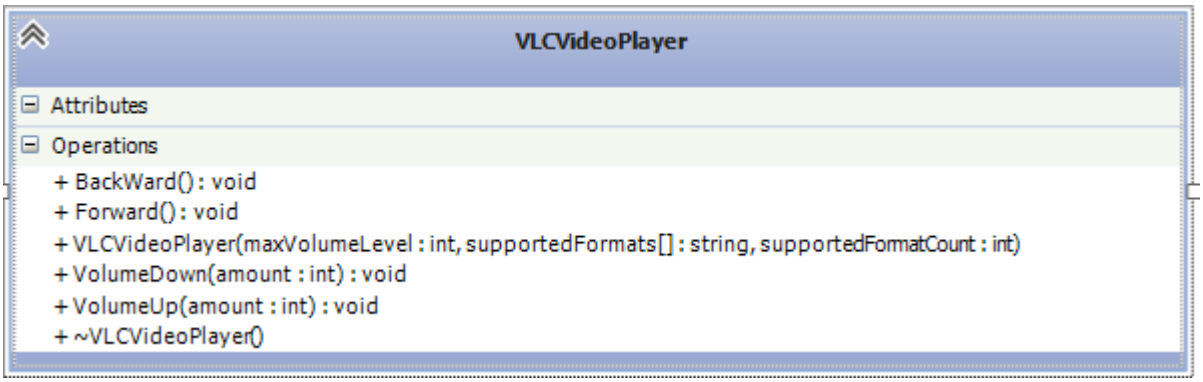

Figure 7:VLCVideoPlayer

Sample Output

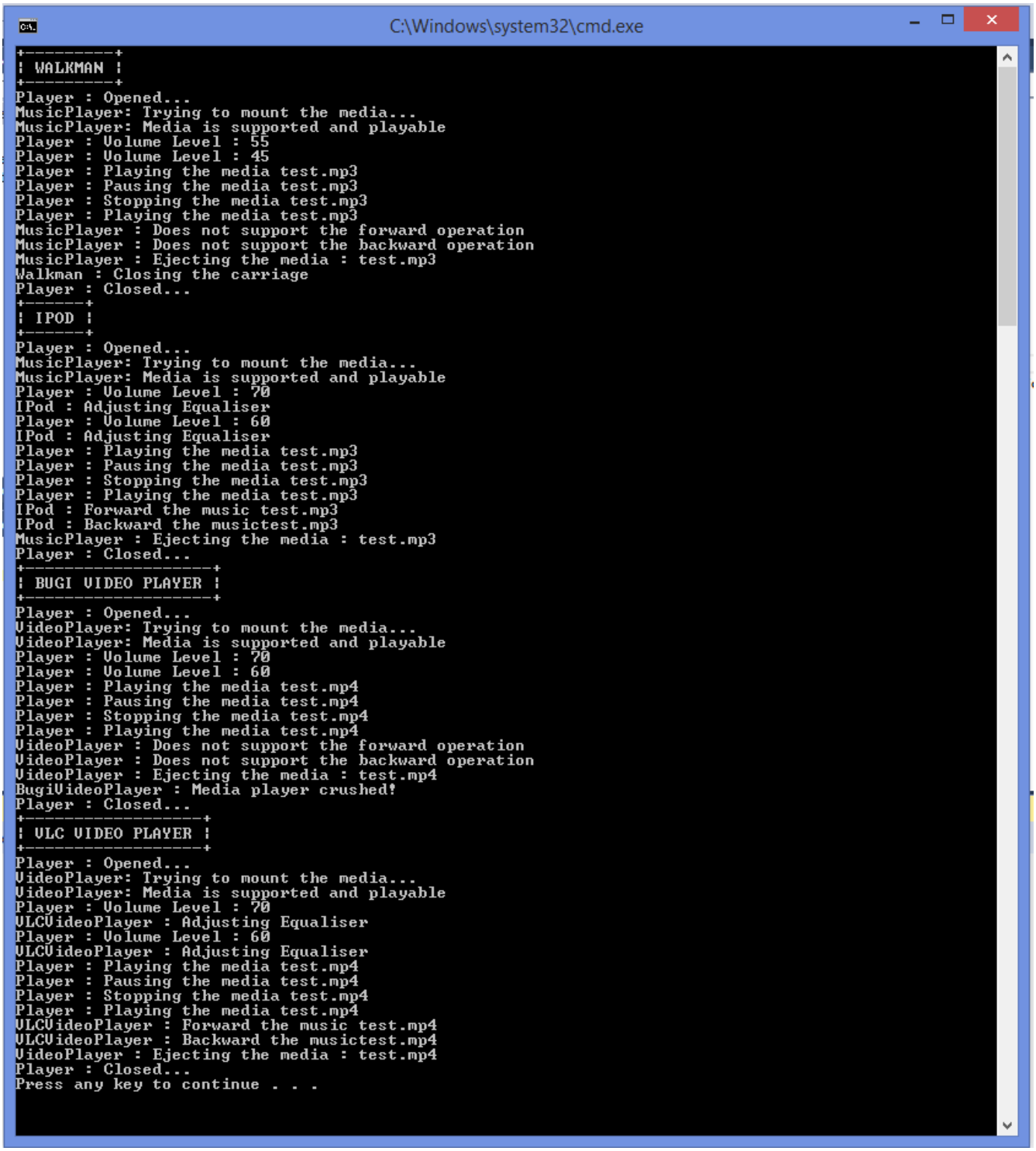

Figure 8:Sample Output

Test Code(You are not allowed to change the test code)

```
/*****************************************
******************************************
* IDE : Visual Studio 2013               *
* Author : Cihan UYANIK                  *
* Experiment 6: Polymorphism             *
******************************************/
#include <iostream>
#include "IPod.h"
#include "Walkman.h"
#include "BugiVideoPlayer.h"
#include "VLCVideoPlayer.h"
using namespace std;

void TEST_ForAll(Player* player,const string& mediaName)
{
	player->Open();
	player->MounthMedia(mediaName);
	player->VolumeUp(20);
	player->VolumeDown(10);
	player->Play();
	player->Pause();
	player->Stop();
	player->Play();

	player->Forward();
	player->BackWard();
	player->EjectMedia();
	player->Close();
}


int main()
{
	cout<< "+---------+" << endl
		<< "| WALKMAN |" << endl
		<< "+---------+" << endl;
	string walkmanFormats[] = { "mp3" };
	Player* walkman = new Walkman(70, walkmanFormats, 1);
	TEST_ForAll(walkman,"test.mp3");

	cout<< "+------+" << endl
		<< "| IPOD |" << endl
		<< "+------+" << endl;
	string iPodFormats[] = { "mp3","wav" };
	MusicPlayer* ipod  = new IPod(100, iPodFormats, 2);
	TEST_ForAll(ipod, "test.mp3");

	cout << "+------------------+" << endl
		<< "| BUGI VIDEO PLAYER |" << endl
		<< "+------------------+" << endl;

	string bugiVideoFormats[] = { "mp4" };
	BugiVideoPlayer* bugiVideoPlayer= new BugiVideoPlayer(100, bugiVideoFormats, 1);
	TEST_ForAll(bugiVideoPlayer, "test.mp4");
```

```
	cout << "+------------------+" << endl
	     << "| VLC VIDEO PLAYER |" << endl
	     << "+------------------+" << endl;

	string vlcVideoFormats[] = { "mp4","avi" };
	Player* vlcVideoPlayer = new VLCVideoPlayer(100, vlcVideoFormats, 2);
	TEST_ForAll(vlcVideoPlayer, "test.mp4");



	return 0;
}
```

Problem-Solving Tips

1- Use UML Diagrams and test code as guide.