Homework 6

Yuji Shimojo

CMSC 330

Instructor: Prof. Reginald Y. Haseltine

July 21, 2013

HOMEWORK 6

Question 1

What is the output of the following C++ program?

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

class Circle
{
public:
   Circle(double radius) {this->radius = radius; }
   void put() const {cout << "Radius = " << radius;}
private:
   double radius;
};

class ColoredCircle: public Circle
{
public:
   ColoredCircle(double radius, string color);
   void put() const;
private:
   string color;
};

ColoredCircle::ColoredCircle(double radius, string color)
```

```
    : Circle(radius), color(color) {}


void ColoredCircle::put() const
{
    Circle::put();
    cout << " Color = " << color;
}


int main()
{
    ColoredCircle redCircle(100., "red");
    Circle* circle1 = &redCircle;
    circle1->put();
    cout << endl;
    Circle circle(50.);
    Circle* circle2 = &circle;
    circle2->put();
    cout << endl;
    return 0;
}
```

Output

```
Radius = 100
Radius = 50
```

Question 2

Modify the program so that the put function is virtual. What is the output after that change?

Answer 2

I changed the line "void put() const {cout << "Radius = " << radius;}" to "virtual void put() const {cout << "Radius = " << radius;}", then I got the following output.

Output

```
Radius = 100 Color = red
Radius = 50
```

Question 3

Does Java allow both virtual and nonvirtual methods? If not, which does it allow? Rewrite this program in Java and identify at least four differences between the programs in the two languages.

Answer 3

Circle and ColoredCircle Classes in Java

```
// Circle.java
public class Circle // Super class
{
	private double radius; // A member variable of super class
	Circle(double radius) // Constructor of super class
	{
		this.radius = radius;
```

```
		}
		public void put()
		{
			System.out.print("Radius = " + radius + " ");
		}
		public static void main(String[] args) // Main method
		{
			// Instantiates a super class object
			Circle circle1 = new ColeredCircle(100., "red");
			circle1.put();
			System.out.println();
			// Instantiates a sub class object
			Circle circle2 = new Circle(50.);
			circle2.put();
			System.out.println();
		}
	}

	// ColeredCircle.java
	public class ColeredCircle extends Circle // Sub class
	{
		private String color;
		ColeredCircle(double radius, String color) // Constructor of sub class
		{
			super(radius);
			this.color = color;
		}
```

```
        @Override
        public final void put() // Overrided put() method
        {
                super.put();
                System.out.print("Color = " + color);
        }
}
```

Output

```
Radius = 100.0 Color = red
Radius = 50.0
```

I identified four differences on inheritance and method overriding between C++ and Java as follows.

	C++	Java
Overriding	You need to add virtual operator to functions expressly.	Basically, methods are virtual by default except for methods with final modifier. If multiple methods with the same name in a superclass and a subclass exist, the method in subclass is automatically overridden.
Modifiers	If you add const modifier to an overriding method in a superclass, you can compile the program by adding const to overridden method in a subclass.	If you add final modifier to overriding method, you cannot override the method in a subclass.
Reference to Superclass	To reference to a superclass in a subclass, you use scope resolution operator.	To reference to a superclass in a subclass, you just use keyword super.

HOMEWORK 6

Calling Overridden Functions/Methods	You can call overridden class member functions through pointer variables.	You can call overridden class member methods through objects of the class.

Question 4

This program contains an example of object slicing. On what line does it occur? Why must it happen?

Answer 4

Object slicing is occurred in the line of "circle1 = redCircle;" in the main method as illustrated in Figure 1 and Figure 2 below. It happens when you assign an object value of subclass to a superclass object.

Figure 1

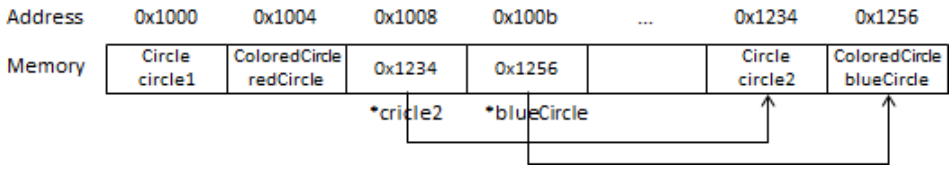
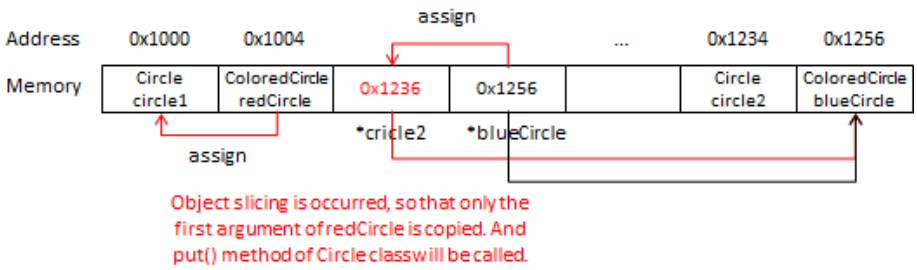


Figure 2



Question 5

HOMEWORK 6

Explain why this never happens in Java. Do some investigating and determine how C# avoids this problem.

Answer 5

Object slicing doesn’t happen in Java because all object variables are references. When you assign a subclass instance to a superclass variable, you just copy the reference.

In C#, using pointers are allowed only in classes or methods with unsafe context. Therefore, programmers can avoid problems due to object slicing unconsciously.