[Homework 6](https://learn.umuc.edu/d2l/lms/dropbox/user/folder_submit_files.d2l?ou=11631&db=10414)

Available Feb 9, 2014 1:00 AM

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;

}

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

Before Change output:

Radius = 100

Radius = 50

After Change output:

Radius = 100 Color = red

Radius = 50

Java natively does virtual methods, you have to explicitly say final to get it to be a nonvirtual method.

Differences:

1. Because java is natively virtual methods you’d have to put the reserved word final on the methods to make them non virtual. So instead of saying virtual you have to/can say final.
2. The ColoredCircle class is has the put inside Colored Circle for C++, but outside and overridden for Java.

C++:

class ColoredCircle: public Circle

{

public:

ColoredCircle(double radius, string color);

virtual void put() const;

private:

string color;

};

Java:

class ColoredCircle extends Circle {

public ColoredCircle(double radius, String color) {

super(radius);

this.color = color;

}

@Override

public void put() {

super.put();

System.out.print(" Color = " + color);

}

private String color;

c. No need for a return statement in the main method for Java

d. The Java version of code needs the qualifiers public/private on the Class names

1. Consider the following C++ main function in place of the one in previous problem.

int main()

{

Circle circle1(100.), \*circle2 = new Circle(200.);

ColoredCircle redCircle(300, "red"), \*blueCircle = newColoredCircle(400., "blue");

circle1 = redCircle;

circle2 = blueCircle;

circle1.put();

cout << endl;

circle2->put();

cout << endl;

return 0;

}

Assuming the put function was made virtual, what is the output of the above program? This program contains an example of object slicing. On what line does it occur? Why must it happen? Explain why this never happens in Java. Do some investigating and determine how C# avoids this problem.

Assuming “newColoredCircle” is supposed to be “new ColoredCircle”

**Answer:**

Radius = 300

Radius = 400 Color = blue

I believe the splice happens when you call put() the first time, you have to call by value because there’s nothing to call by reference. That’s why when you call by reference the second put() it works fine.

In Java all methods are natively virtual, and so the methods can be overridden. C# avoids the problem by not permitting call by-value for reference types.