Solution to Practice Problems: Classes, Inheritance, and Polymorphism

Here's a problem from a previous Final Exam.

Consider the following skeleton for a Robot class, which has private fields for storing the location of a Robot object, its name, and the direction it’s facing (North for a direction parallel to the positive y axis, South for the negative y axis, East for the positive x axis, or West for the negative x axis). It also has stub methods for constructing a Robot object, changing the direction, and moving the location of the robot in the direction it’s facing.

```java
public class Robot {
    private String name;
    private char direction; // 'N', 'S', 'E', or 'W'
    private int xLoc, yLoc; // the (x, y) location of the robot

    // Initialize name, direction, and (x, y) location
    public Robot(String name, char dir, int x, int y) { ... }

    public String toString() {
        return name + " is standing at (" + x + "," + y + ") and facing " + direction);
    }

    // turn 90 degrees clockwise, e.g. 'N' changes to 'E', 'E' to 'S', ...
    public void turnClockwise() { ... }

    // turn 90 degrees counterclockwise, e.g. 'N' to 'W', 'W' to 'S', ...
    public void turnCounterClockwise() { ... }

    // move numSteps in direction you are facing,
    // e.g. if 'N' 3 steps, then y increases 3
    public void takeSteps(int numSteps) { ... }
}
```
(a) Assuming the class above is completed correctly, what does the following program display on the screen:

```java
public static void main(String args[])
{
    Robot robby = new Robot("Robby", 'N', 10, 12);
    for (int i = 0; i < 5; i++)
    {
        if (i % 2 == 0)
        {
            robby.turnClockwise();
        }
        else
        {
            robby.turnCounterClockwise();
        }
        robby.takeSteps(3);
        System.out.println(robby);
    }
}
```

Displayed on screen:

Robby is standing at (13, 12) and facing E
Robby is standing at (13, 15) and facing N
Robby is standing at (16, 15) and facing E
Robby is standing at (16, 18) and facing N
Robby is standing at (19, 18) and facing E

(b) Complete the constructor, the turnClockwise method, and the takeSteps method. You do not need to define turnCounterClockwise.

```java
public Robot(String name, char dir, int x, int y)
{
    this.name = name;
    this.direction = dir;
    this.xLoc = x;
    this.yLoc = y;
}

public void turnClockwise()
{
    if(direction=='N') { direction = 'E'; } 
    else if(direction=='E') { direction = 'S'; } 
    else if(direction=='S') { direction = 'W'; } 
    else { direction = 'N'; } 
}

public void takeSteps(int numSteps)
{
    if(direction=='N') { yLoc += numSteps; } 
    else if(direction=='E') { xLoc += numSteps; } 
    else if(direction=='S') { yLoc -= numSteps; } 
```
else { xLoc -= numSteps; }
}

(c) Write Java code to create an array of 5 robots. Use a for loop to fill in the array so that the n-th robot is named "robot n", and it starts off life facing east at location (n, n).

```java
Robot [] robots = new Robot[5];
for(int i=0; i<robots.length; i++)
{
    robots[i] = new Robot("robot " + i, 'E', i, i);
}
```

Here's another problem from a previous Final Exam. This one is an inheritance/polymorphism question.

```java
class SuperClass
{
    protected int x = 0;

    public SuperClass(int x)
    {
        this.x = x;
    }

    private void increment() { x++; }

    protected final void add(int y)
    {
        x += y;
    }

    public void display()
    {
        System.out.println(x);
    }
}

public class SubClass extends SuperClass
{
    public SubClass(int x)
    {
        super(x);
    }

    public void display()
    {
        add(2);
        super.display();
    }
```
```java
public static void main(String[] args) {
    SuperClass sc = new SuperClass(3);
    sc.display();

    sc = new SubClass(3);
    sc.display();
}
```

(a) List the name of all methods that subclasses of SuperClass inherit.

*subclasses inherit all methods: the constructor, increment, add, and display.* If you want, you could also list all of the methods that SuperClass implicitly inherits from the Object class (e.g., equals, toString, etc.), but that's not required.

(b) List the name of all methods that are visible in subclasses of SuperClass (in other words, methods that can be called directly).

*add can be called directly just by using the name add(). The constructor SuperClass can be called by using the super() constructor. The display() method from SuperClass is overridden by the display() method in the SubClass, but it can still be called by writing super.display(). In summary, any method that has public or protected access in the superclass can be called directly by the subclass.*

(c) List the name of all methods that may NOT be overridden by any subclasses of SuperClass.

*methods that are declared to be final in the superclass may not be overridden. So the add() method may not be overridden.*

(d) What gets displayed on the screen when SubClass is executed?

**displayed on screen:**

```
3
5
```
| public class Alpha { |
| private int a = 0; |
| public void alpha() { |
| a++; |
| System.out.println(a); |
| } |
| public static void main |
| (String [] args) { |
| Alpha a = new Alpha(); |
| a.alpha(); |
| } |
| } |
| public class Beta |
| extends Alpha |
| { |
| protected int a = 0; |
| public Beta(int x) { |
| this.a = x; |
| } |
| public void beta() { |
| a+=2; |
| System.out.println(a); |
| } |
| public static void main |
| (String [] args) { |
| Alpha a = new Alpha(); |
| a.alpha(); |
| Beta b = new Beta(2); |
| b.alpha(); |
| b.beta(); |
| } |
| } |
| public class Nu |
| extends Beta |
| { |
| public Nu() { |
| super(10); |
| } |
| public void nu() { |
| a+=5; |
| System.out.println(a); |
| } |
| public static void main |
| (String [] args) { |
| Nu n = new Nu(); |
| n.nu(); |
| n.beta(); |
| n.alpha(); |
| Beta b = new Beta(10); |
| b.alpha(); |
| b.beta(); |
| } |
| } |

Output from executing the Alpha class:
1

Output from executing the Beta class:
1
1
4

Output from executing the Nu class:
15
17
1
1
12