Inheritance of Existing Classes

Inheritance is a way to reuse existing code. It involves creating a new class that inherits from an existing class (the base class).

### Example:

```java
public class DrunkTurtle extends Turtle {
    /**
     * A drunk turtle is a turtle that "staggers" as it moves forward.
     */
    public void moveForward(double units) {
        for (int i = 0; i < units; i++) {
            if (Math.random() < 0.1) {
                turnLeft(((int) (Math.random() * 60) - 30));
            }
            super.moveForward(1);
        }
    }
}
```

### Design Considerations:

- **Visibility:** The protected visibility allows controlled access to the base class components.
- **Access:** Use protected access for components you want to make accessible to derived classes.
- **Client Awareness:** Avoid increasing client awareness of the base class components.

### Implementation Tips:

- **Beans:** Use `protected` for fields to control access.
- **Methods:** Use `public` for methods that can be accessed by clients.
- **Hiding:** Use `private` for fields that are not intended to be accessed or modified from outside the class.
- **Encapsulation:** Ensure that the class provides a clear interface.

### Summary:

Inheritance allows for code reuse and promotes better design practices by promoting modularity and abstraction.
בשפת Java

class CartesianPoint

public void translate(double dx, double dy) {
    x += dx;
    y += dy;
}

public void translate(double dX, double dY) {
    double newX = x + dX;
    double newY = y + dY;
    x = Math.atan2(newY - y, newX - x);
    y = Math.sqrt(newX * newX + newY * newY);
}

public static double distance(CartesianPoint ap1, CartesianPoint ap2) {
    double dx = ap1.x - ap2.x;
    double dy = ap1.y - ap2.y;
    return Math.sqrt(dx * dx + dy * dy);
}

class PolarPoint

public void translate(double dx, double dy) {
    r = Math.sqrt(r * r + dx * dx + dy * dy);
    theta = Math.atan2((y + dy) * Math.sin(theta) + (x + dx) * Math.cos(theta), r * Math.cos(theta) - (y + dy) * Math.sin(theta) - (x + dx) * Math.cos(theta));
}

public static double distance(PolarPoint pp1, PolarPoint pp2) {
    double dx = pp1.r - pp2.r;
    double dy = pp1.theta - pp2.theta;
    return Math.sqrt(dx * dx + dy * dy);
}

class SmartPoint

private double x;
private double y;
public CartesianPoint(double x, double y) {
    this.x = x;
    this.y = y;
}

public double x() {
    return x;
}

public double y() {
    return y;
}

public double rho() {
    return Math.sqrt(x * x + y * y);
}

public double theta() {
    return Math.atan2(y, x);
}

private double r;
private double theta;
public PolarPoint(double r, double theta) {
    this.r = r;
    this.theta = theta;
}

public double x() {
    return r * Math.cos(theta);
}

public double y() {
    return r * Math.sin(theta);
}

public double rho() {
    return r;
}

public double theta() {
    return theta;
}

public void rotate(double angle) {
    double currentTheta = Math.atan2(y, x);
    double currentRho = rho();
    x = currentRho * Math.cos(currentTheta + angle);
    y = currentRho * Math.sin(currentTheta + angle);
}

public void translate(double dx, double dy) {
    x += dx;
    y += dy;
}

public static double distance(SmartPoint sp1, SmartPoint sp2) {
    double dx = sp1.x() - sp2.x();
    double dy = sp1.y() - sp2.y();
    return Math.sqrt(dx * dx + dy * dy);
}
public class CartesianPoint {
    public double distance(IPoint other) {
        double deltaX = x() - other.x();
        double deltaY = y() - other.y();
        return Math.sqrt(deltaX * deltaX + deltaY * deltaY);
    }
}

public class PolarPoint {
    public double distance(IPoint other) {
        double deltaX = x() - other.x();
        double deltaY = y() - other.y();
        return Math.sqrt(deltaX * deltaX + deltaY * deltaY);
    }
}

Method Overloading & Overriding

public class A {
    public float foo(float a, float b) {...}
}

public class B extends A {
    public float foo(int a) {...}
}

Which of the following methods can be defined in B:
1. float foo(float a, float b){...}
2. public float foo(float p, float q) {...}
3. public int foo(int a) {...}

✓ 2. public float foo(float p, float q) {...}
✓ 3. public int foo(int a) {...}
Method Overriding

```java
public class A {
    public void print() {
        System.out.println("A");
    }
}

public class B extends A {
    public void print()
    { System.out.println("B"); }
}

public class C {
    public static void main(...) {
        B b = new B();
        A a = b;
        b.print();
        a.print();
    }
}
```

What is the output?

The output is:

```
B
B
```
בתחום של תרגיל 7
האם טרור ליציץ בדיקת אוכל? 
איך דואגים שיהיה בדיוק_instance יחידי? 
Singletone design pattern