Encapsulation. Use accessor methods, not public fields

```
//Classes like this are a sign of bad design
class Point {
	public double x;
	public double y;
}
```

Because the data fields of such classes are accessed directly, these classes **do not offer the benefits of encapsulation.** You can't change the representation without changing the API, you can't enforce invariants, and you can't take auxiliary action when a field is accessed. These classes also go against the principles of Object-Oriented programming and should, in 99.99% of situations, be replaced by classes with private fields and public accessor methods (getters) and, for mutable classes, mutators (setters):

How to solve:

```
//Encapsulation of data by accessor methods and mutators
class Point {
private double x;
private double y;

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

public double getX() { return x; }
public double getY() { return y; }
public void setX(double x) { this.x = x; }
public void setY(double y) { this.y = y; }
}
```

If a public class exposes its data fields, all hope of changing its representation is lost, as client code can be distributed far and wide.