S.O.L.I.D

קווים מנחים לעיצוב קוד נכון

סימפטומים של קוד לא מתוכנן טוב
AKA – קוד ספגטי

- כל שינוי בקוד משפיע על **הרבה** חלקים בקוד.
- שינוי בקוד משפיע על אזורים **לא קשורים** בקוד.
- קוד **לא פריק**. לא ניתן להשתמש בקוד שכבר כתבנו בהקשרים אחרים מאלו שלשמם נכתב הקוד במקור.

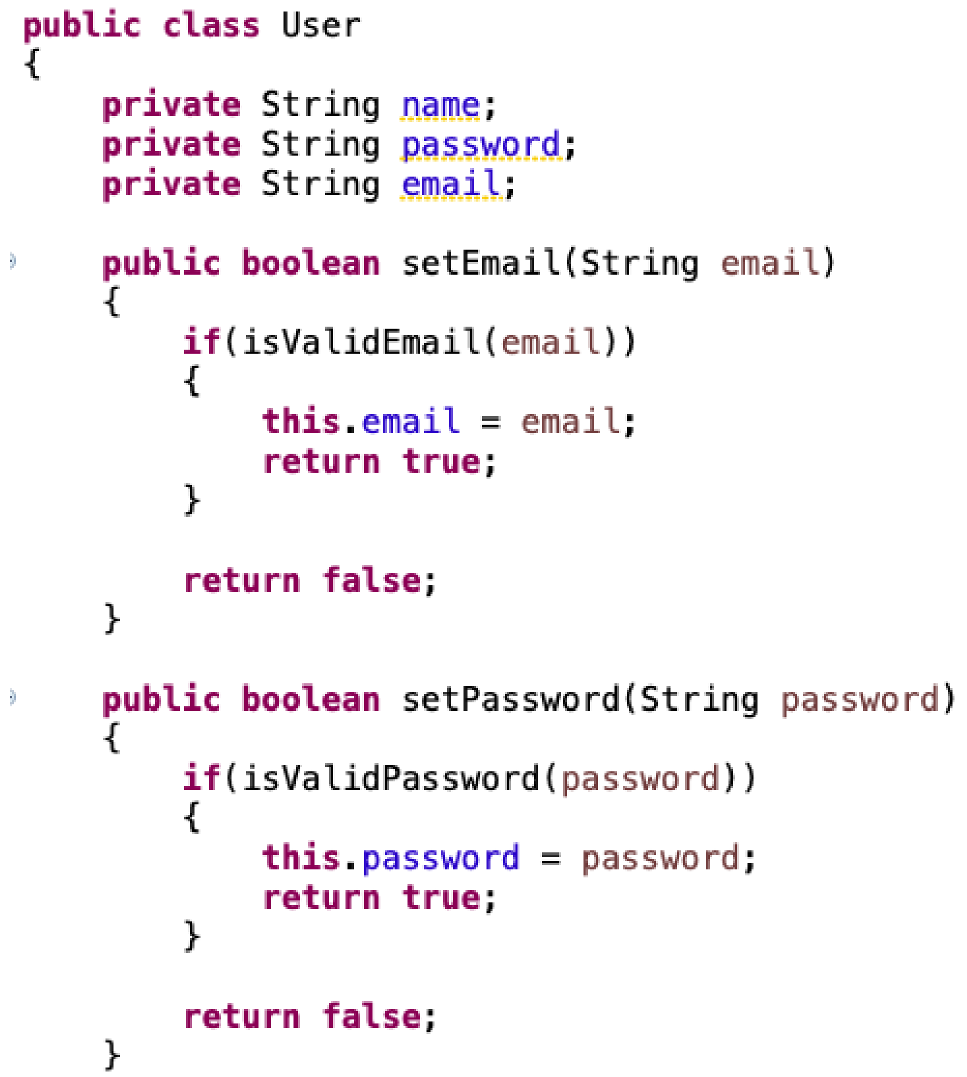
האופי המרכזי של הבעיות האלו הוא יותר מידי תלות בתוך הקוד.

עקרונות SOLID באים לתת קווים מנחים שיגרמו לנו להימנע מלכתוב קוד עם הבעיות הנ"ל.

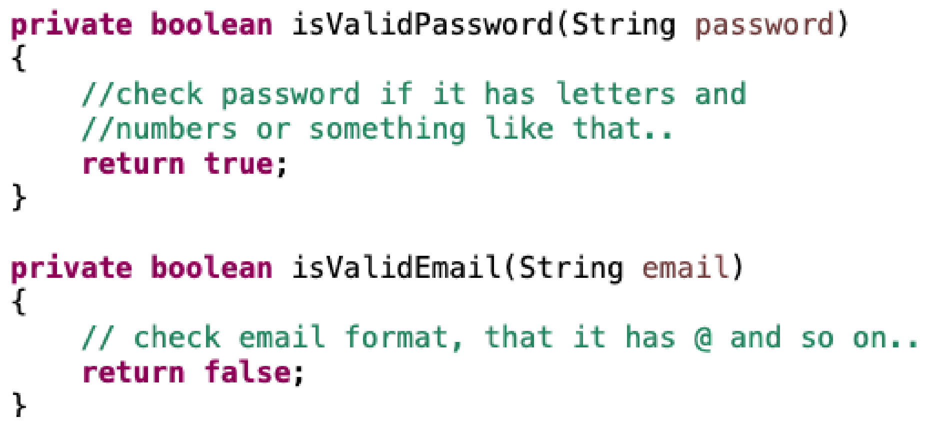
S - Single Responsibility Principle

למחלקה צריך להיות תחום אחריות **אחד**

Bad:

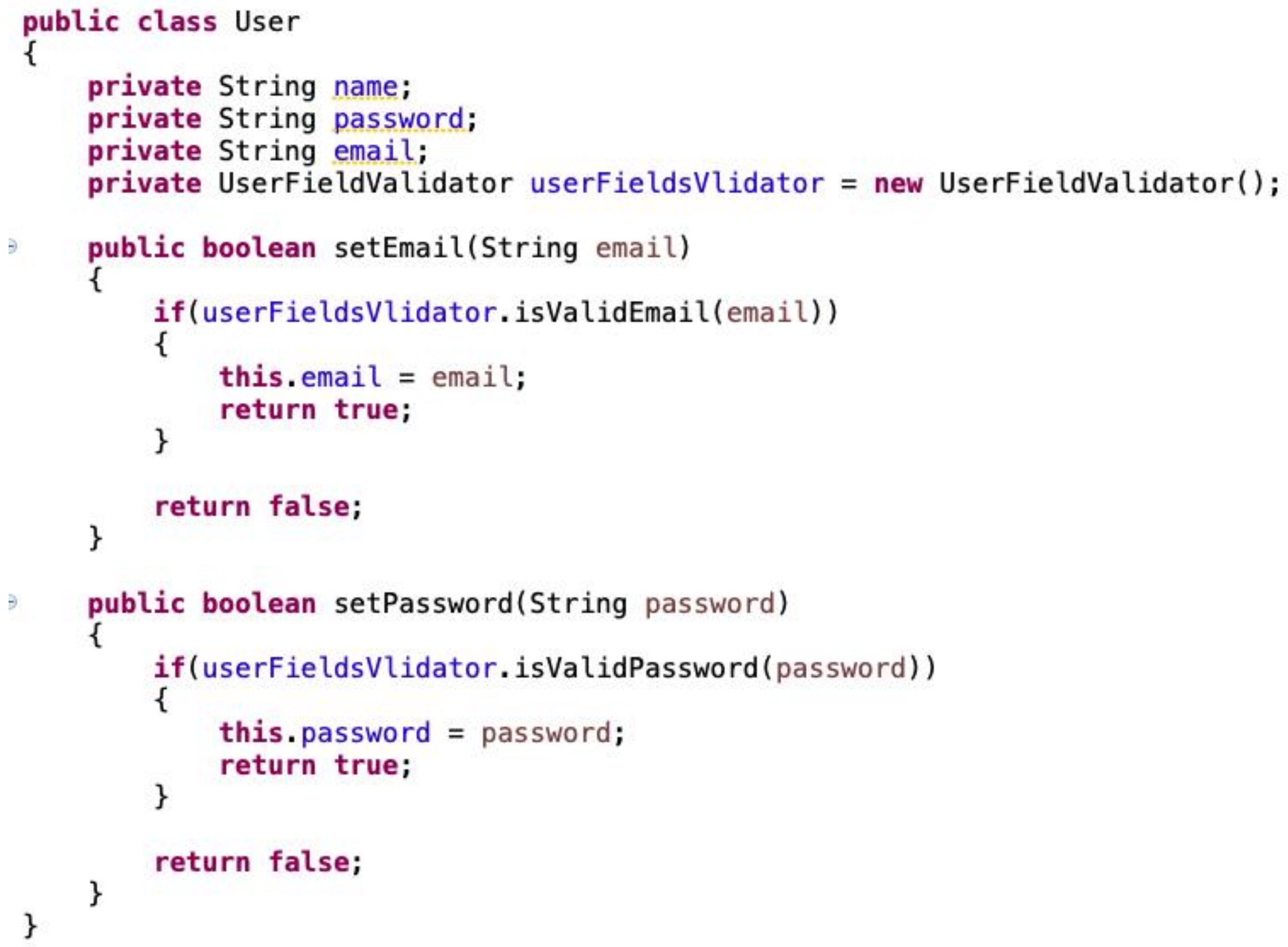


```java
public class User
{
    private String name;
    private String password;
    private String email;

    public boolean setEmail(String email)
    {
        if(isValidEmail(email))
        {
            this.email = email;
            return true;
        }

        return false;
    }

    public boolean setPassword(String password)
    {
        if(isValidPassword(password))
        {
            this.password = password;
            return true;
        }

        return false;
    }
```

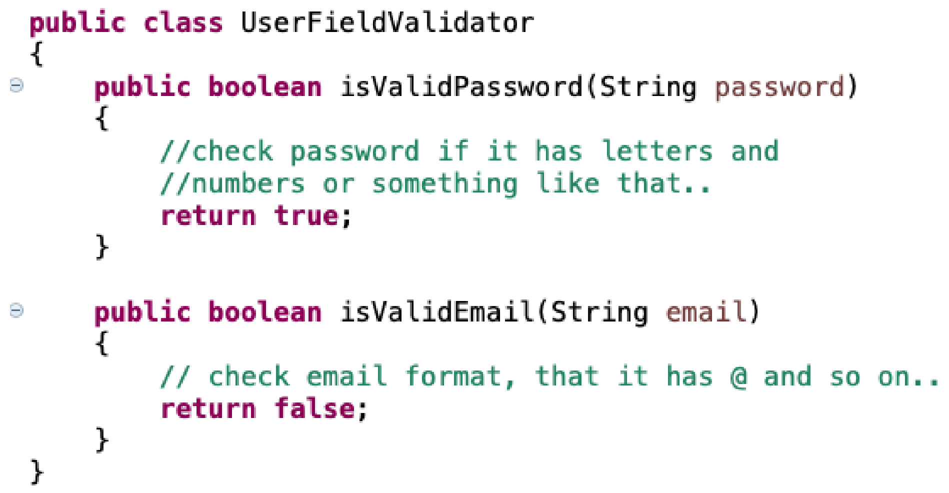


```java
private boolean isValidPassword(String password)
{
    //check password if it has letters and
    //numbers or something like that..
    return true;
}

private boolean isValidEmail(String email)
{
    // check email format, that it has @ and so on..
    return false;
}
```

Good:



```java
public class User
{
    private String name;
    private String password;
    private String email;
    private UserFieldValidator userFieldsVlidator = new UserFieldValidator();

    public boolean setEmail(String email)
    {
        if(userFieldsVlidator.isValidEmail(email))
        {
            this.email = email;
            return true;
        }

        return false;
    }

    public boolean setPassword(String password)
    {
        if(userFieldsVlidator.isValidPassword(password))
        {
            this.password = password;
            return true;
        }

        return false;
    }
}
```

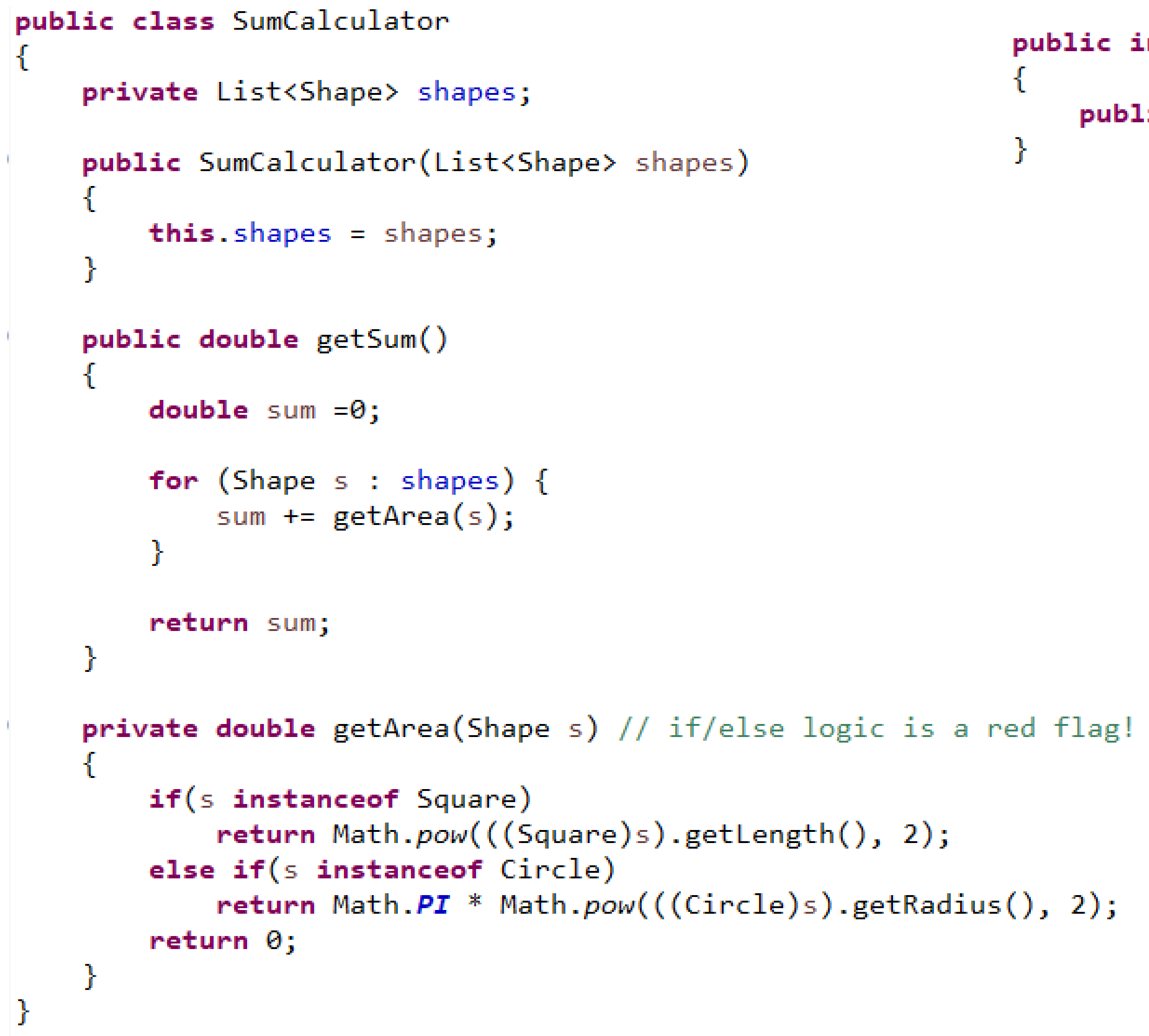


```java
public class UserFieldValidator
{
    public boolean isValidPassword(String password)
    {
        //check password if it has letters and
        //numbers or something like that..
        return true;
    }

    public boolean isValidEmail(String email)
    {
        // check email format, that it has @ and so on..
        return false;
    }
}
```

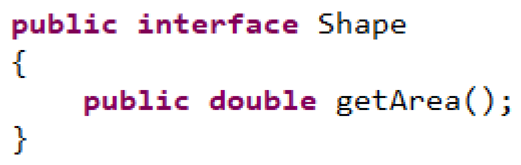
O - Open/Closed Principle

מחלקה צריכה להיות **פתוחה** להוספות **וסגורה** לשינויים

Bad:

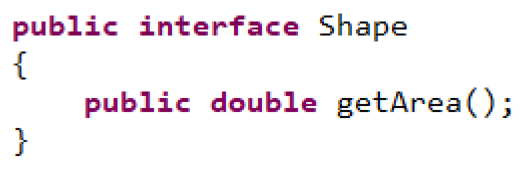


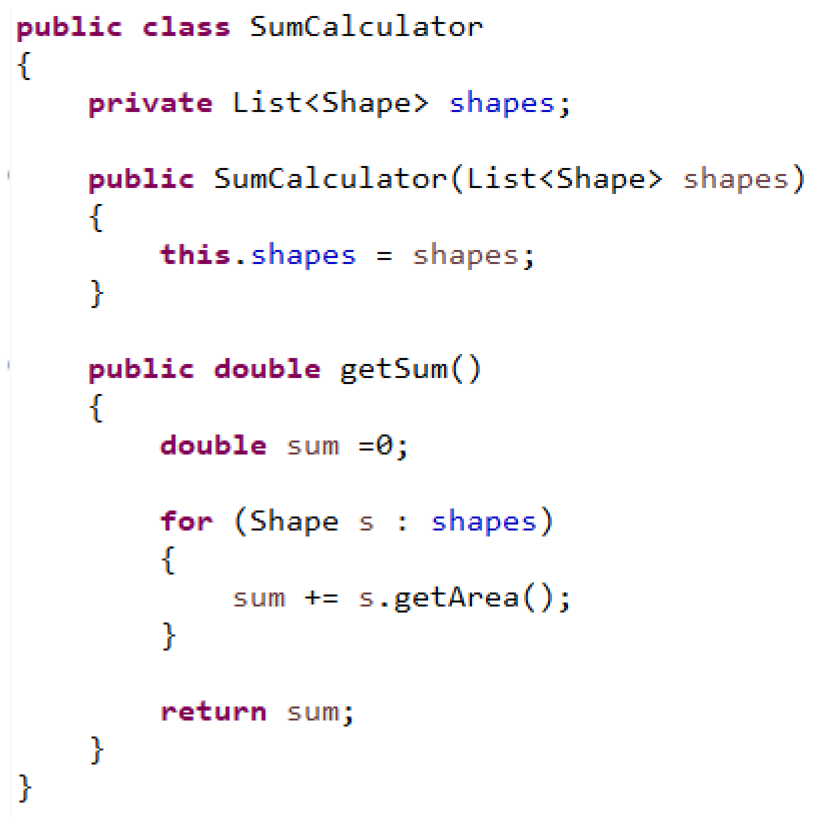
```java
public class SumCalculator
{
    private List<Shape> shapes;

    public SumCalculator(List<Shape> shapes)
    {
        this.shapes = shapes;
    }

    public double getSum()
    {
        double sum =0;

        for (Shape s : shapes) {
            sum += getArea(s);
        }

        return sum;
    }

    private double getArea(Shape s) // if/else logic is a red flag!
    {
        if(s instanceof Square)
            return Math.pow(((Square)s).getLength(), 2);
        else if(s instanceof Circle)
            return Math.PI * Math.pow(((Circle)s).getRadius(), 2);
        return 0;
    }
}
```

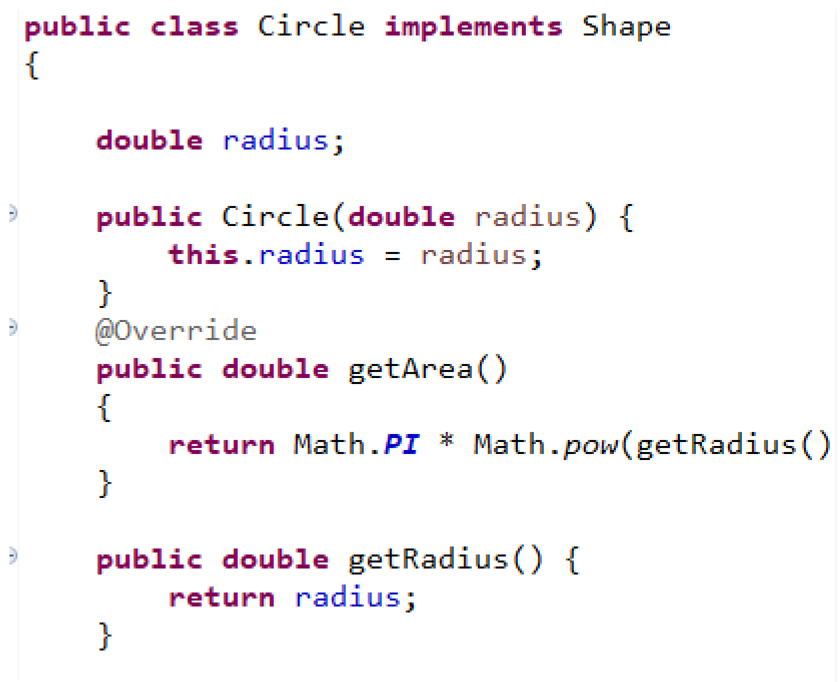


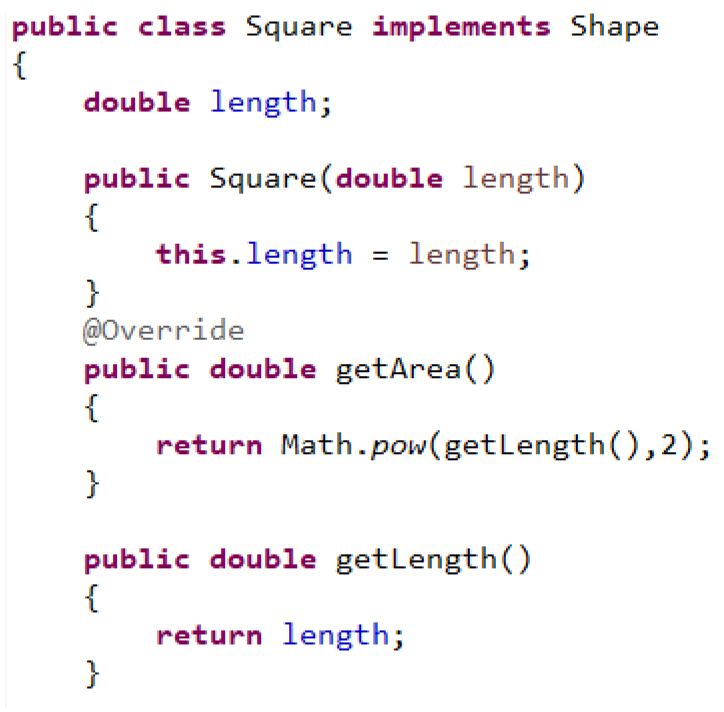
```java
public interface Shape
{
    public double getArea();
}
```

Good:


```java
public interface Shape
{
    public double getArea();
}
```


```java
public class SumCalculator
{
    private List<Shape> shapes;

    public SumCalculator(List<Shape> shapes)
    {
        this.shapes = shapes;
    }

    public double getSum()
    {
        double sum =0;

        for (Shape s : shapes)
        {
            sum += s.getArea();
        }

        return sum;
    }
}
```


```java
public class Circle implements Shape
{

    double radius;

    public Circle(double radius) {
        this.radius = radius;
    }
    @Override
    public double getArea()
    {
        return Math.PI * Math.pow(getRadius()
    }

    public double getRadius() {
        return radius;
    }
```


```java
public class Square implements Shape
{
    double length;

    public Square(double length)
    {
        this.length = length;
    }
    @Override
    public double getArea()
    {
        return Math.pow(getLength(),2);
    }

    public double getLength()
    {
        return length;
    }
```

L - Liskov Substitution Principle

פונקציות המשתמשות במשתנים מסוג מחלקת אב, חייבות להיות מסוגלות לפעול בצורה תקינה גם על כל סוגי האובייקטים מסוג הבן, מבלי להיות מודעות לסוג האובייקט בפועל

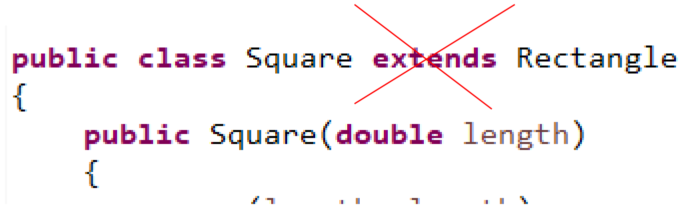
Bad:

```java
public class Rectangle implements Shape
{
    private double width;
    private double height;

     public Rectangle(double width, double height)
     {
         this.width = width;
         this.height = height;
     }

    @Override
    public double getArea()
    {
        return width * height;
    }

    public void setWidth(double width)
    {
        this.width = width;
    }

    public void setHeight(double height)
    {
        this.height = height;
    }
}
```

```java
public class Square extends Rectangle
{
    public Square(double length)
    {
        super(length, length);
    }

    public void setWidth(double width)
    {
        super.setWidth(width);
        super.setHeight(width);
    }

    public void setHeight(double height)
    {
        super.setHeight(height);
        super.setWidth(height);
    }
}
```

```java
public static void foo(Rectangle r)
{
    r.setWidth(2);
    r.setHeight(3);

    // some logic that based on the fact the area is 6
}
```

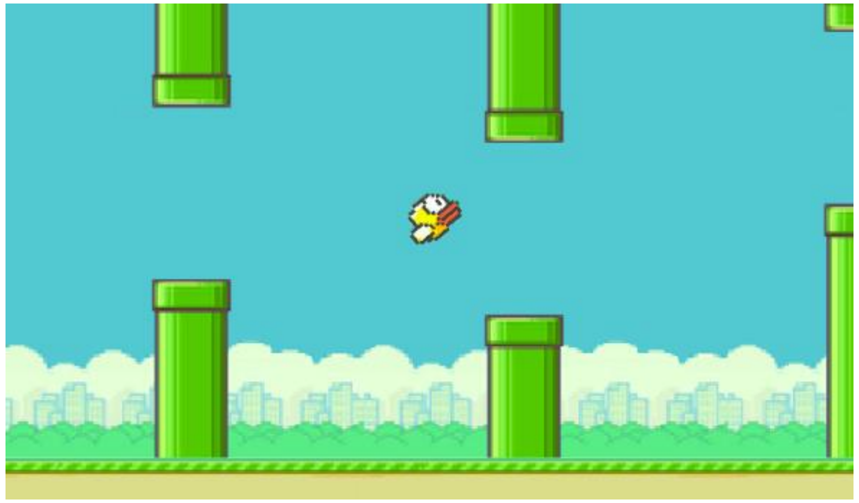
Good:



```
public class Square ~~extends~~ Rectangle
{
    public Square(double length)
    {
```

Bad:

```
public interface Bird
{
    public void setLocation(double latitude, double longitude);
    public void setAltitude(double altitude);
}
```

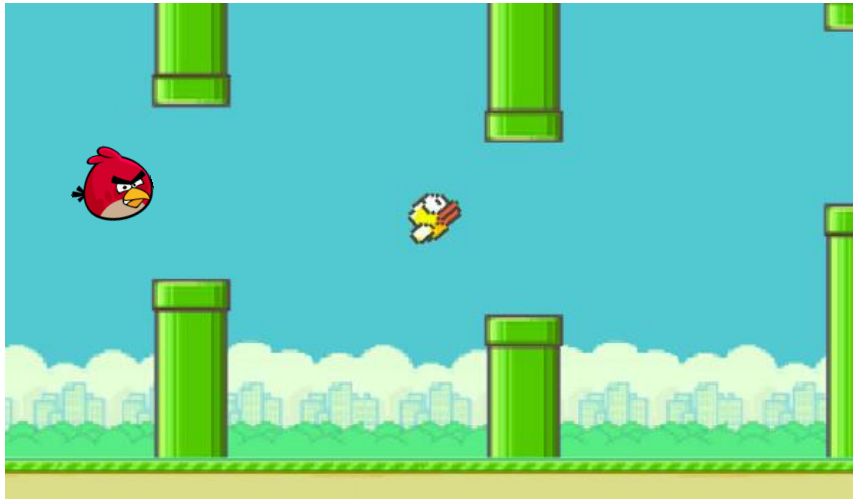


```
public class FlappyBird implements Bird
{
    private double latitude;
    private double logtitude;
    private double altitude;

    @Override
    public void setLocation(double latitude, double longitude)
    {
        this.latitude = latitude;
        this.logtitude = longitude;
    }

    @Override
    public void setAltitude(double altitude)
    {
        this.altitude=altitude;
    }
```

```
public class Game
{
    public void changeHeight(Bird bird, double altitude)
    {
        bird.setAltitude(altitude);
    }
```

Bad:

```java
public interface Bird
{
    public void setLocation(double latitude, double longitude);
    public void setAltitude(double altitude);
}
```



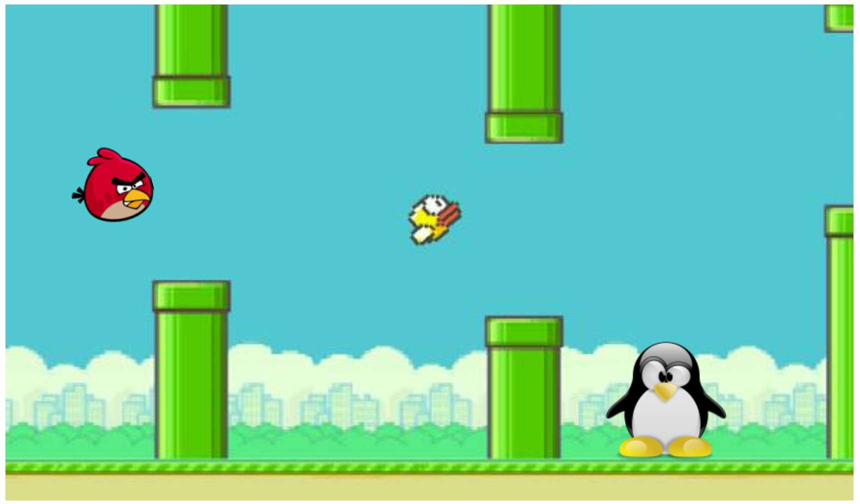
```java
public class FlappyBird implements Bird
{
    private double latitude;
    private double logtitude;
    private double altitude;

    @Override
    public void setLocation(double latitude, double longitude)
    {
        this.latitude = latitude;
        this.logtitude = longitude;
    }

    @Override
    public void setAltitude(double altitude)
    {
        this.altitude=altitude;
    }
```

```java
public class Game
{
    public void changeHeight(Bird bird, double altitude)
    {
        bird.setAltitude(altitude);
    }
```

```java
public class AngryBird implements Bird
{
    //....
```

Bad:

```
public interface Bird
{
    public void setLocation(double latitude, double longitude);
    public void setAltitude(double altitude);
}
```



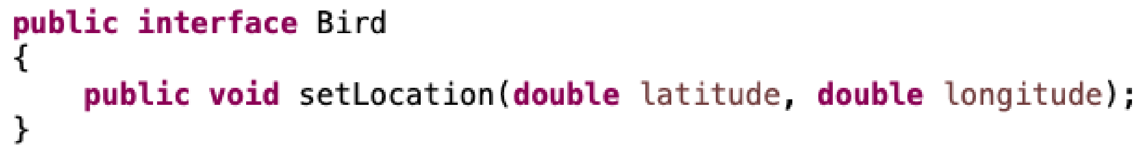
```
public class FlappyBird implements Bird
{
    private double latitude;
    private double logtitude;
    private double altitude;

    @Override
    public void setLocation(double latitude, double longitude)
    {
        this.latitude = latitude;
        this.logtitude = longitude;
    }

    @Override
    public void setAltitude(double altitude)
    {
        this.altitude=altitude;
    }
```

```
public class Game
{
    public void changeHeight(Bird bird, double altitude)
    {
        bird.setAltitude(altitude);
    }
```

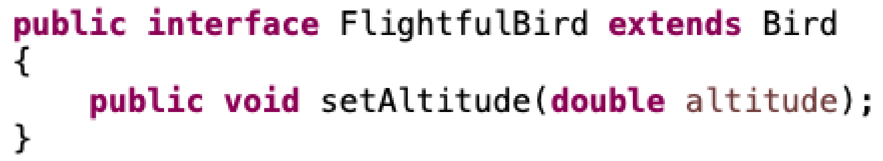
```
public class Penguin implements Bird
{
    private double latitude;
    private double logtitude;

    @Override
    public void setLocation(double latitude, double longitude)
    {
        this.latitude = latitude;
        this.logtitude = longitude;
    }

    @Override
    public void setAltitude(double altitude)
    {
        //nothing to do here ..
    }
}
```

```
public class AngryBird implements Bird
{
    //....
```

Good:



```java
public interface Bird
{
    public void setLocation(double latitude, double longitude);
}
```



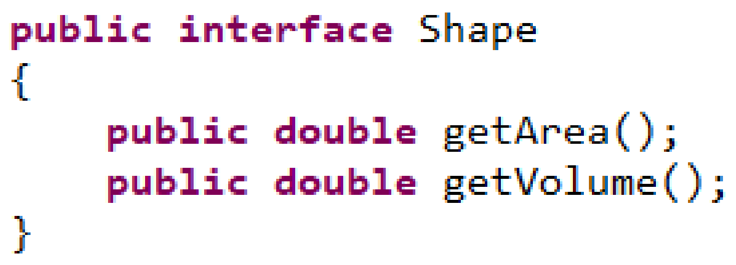
```java
public interface FlightfulBird extends Bird
{
    public void setAltitude(double altitude);
}
```

I- Interface Segregation Principle

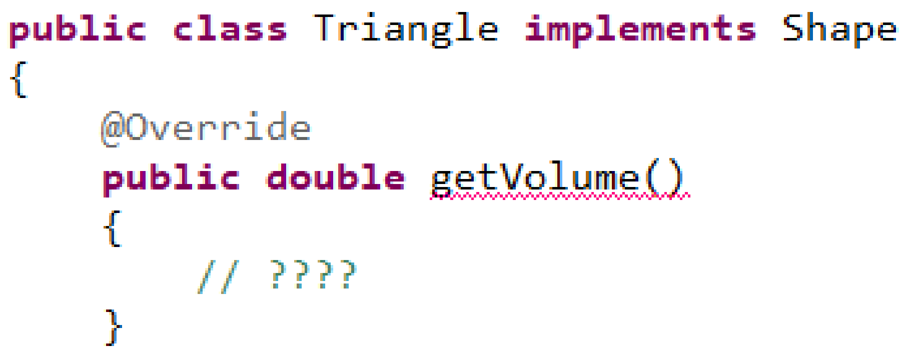
יש לדאוג לממשקים מצומצמים:

- לא לאלץ למחלקה לממש ממשק שאין לה צורך מלא בו.
- לדאוג לכימוס מרבי של מידע.

Bad:

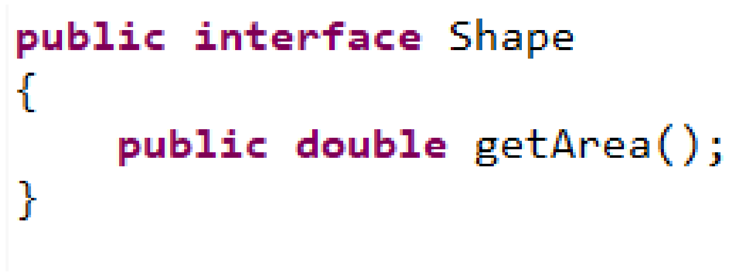


```
public interface Shape
{
    public double getArea();
    public double getVolume();
}
```

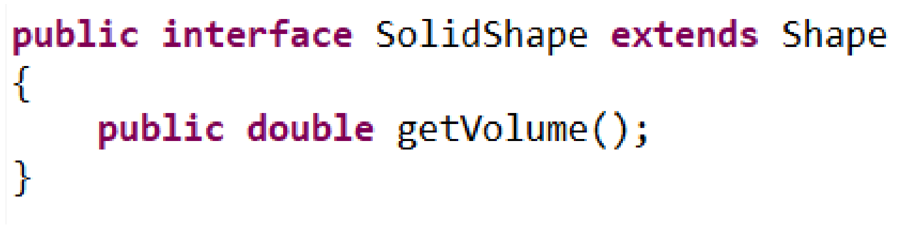


```
public class Triangle implements Shape
{
    @Override
    public double getVolume()
    {
        // ????
    }
```

Good:

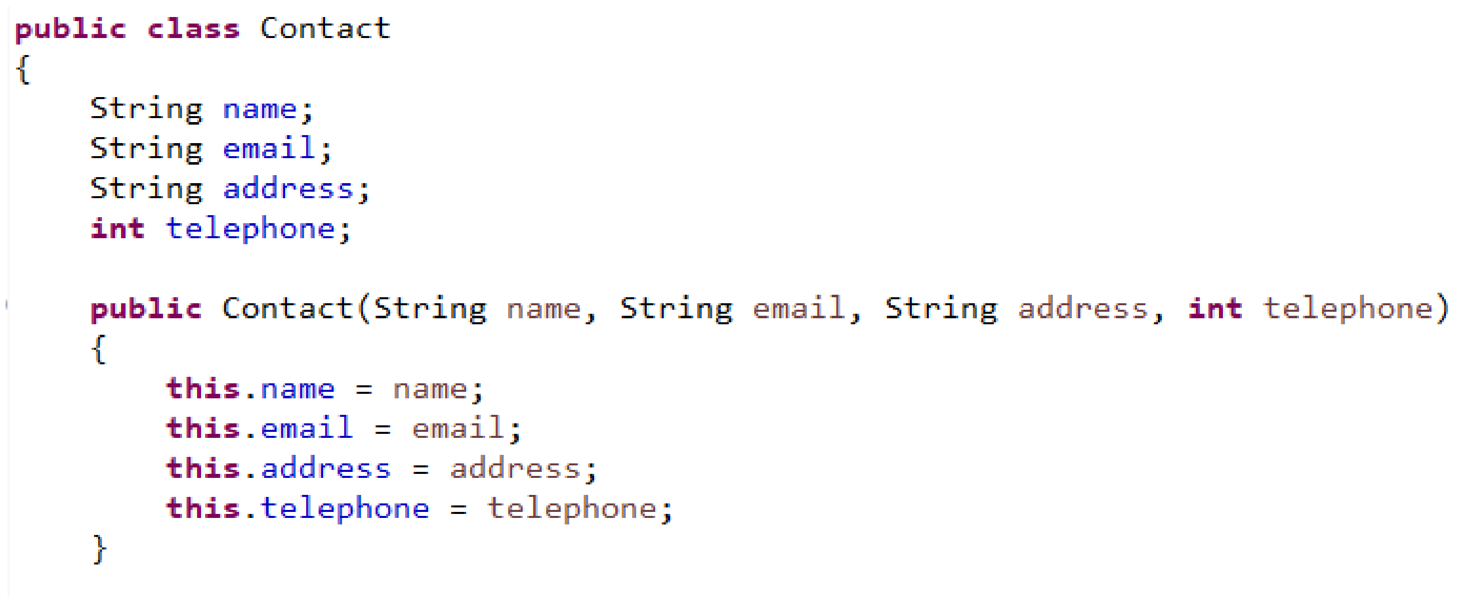


```
public interface Shape
{
    public double getArea();
}
```

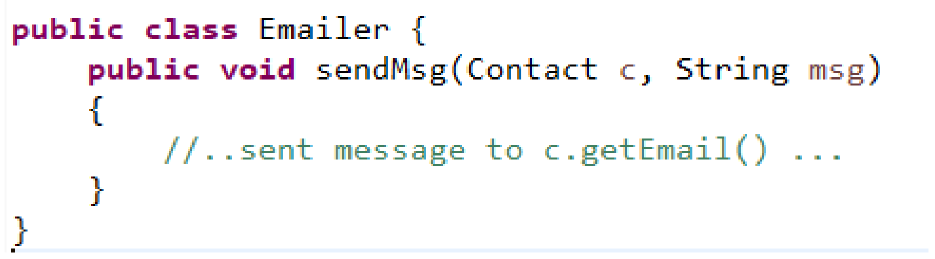


```
public interface SolidShape extends Shape
{
    public double getVolume();
}
```

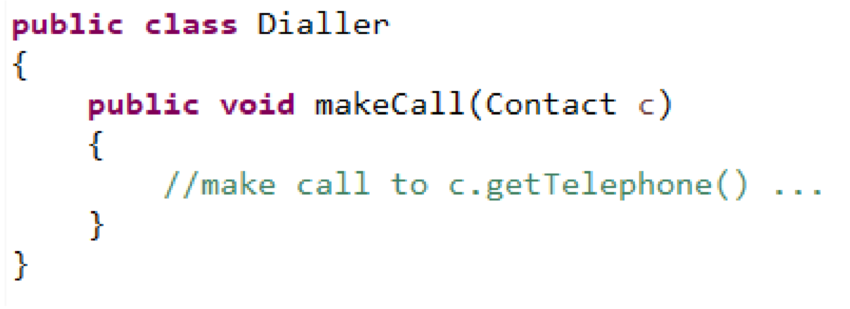
Bad:



```java
public class Contact
{
    String name;
    String email;
    String address;
    int telephone;

    public Contact(String name, String email, String address, int telephone)
    {
        this.name = name;
        this.email = email;
        this.address = address;
        this.telephone = telephone;
    }
```



```java
public class Emailer {
    public void sendMsg(Contact c, String msg)
    {
        //..sent message to c.getEmail() ...
    }
}
```



```java
public class Dialler
{
    public void makeCall(Contact c)
    {
        //make call to c.getTelephone() ...
    }
}
```

Good:

```java
public interface IEmailable
{
    public String getEmail();
}
```

```java
public interface IDiallable
{
    public String getTelephone();
}
```

```java
public class Contact implements IEmailable, IDiallable
{
    String name;
    String email;
    String address;
    int telephone;

    public Contact(String name, String email, String address, int telephone)
    {
        this.name = name;
        this.email = email;
        this.address = address;
        this.telephone = telephone;
    }
```

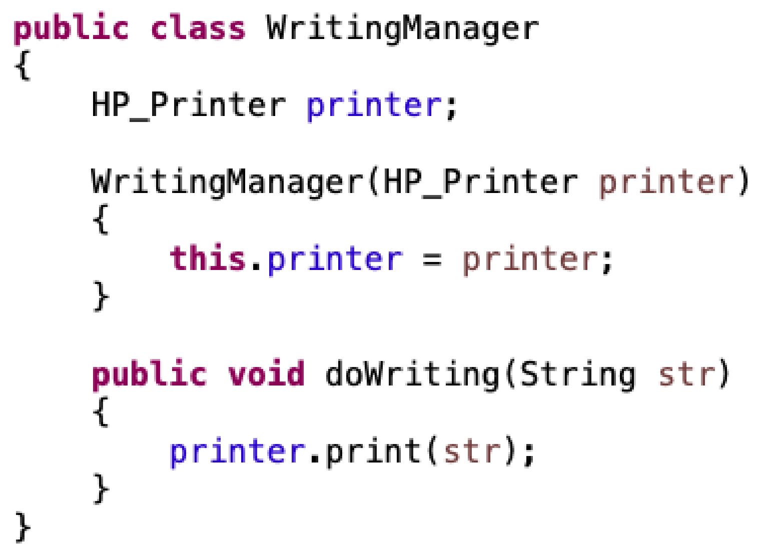
```java
public class Emailer {
    public void sendMsg(IEmailable c, String msg)
    {
        //..sent message to c.getEmail() ...
    }
}
```

```java
public class Dialler
{
    public void makeCall(IDiallable c)
    {
        //make call to c.getTelephone() ...
    }
}
```

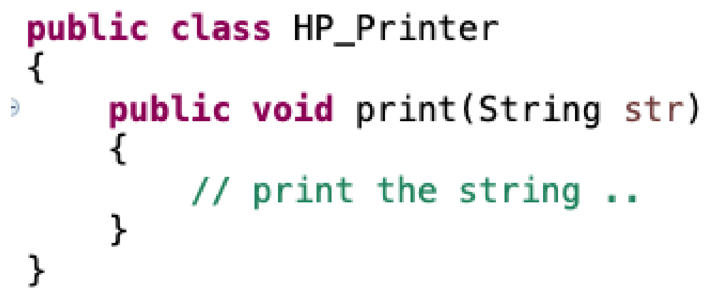
D- Dependency Inversion Principle

מחלקות height level לא צריכות להשתמש באופן ישיר במחלקות low level

Bad:



```java
public class WritingManager
{
    HP_Printer printer;

    WritingManager(HP_Printer printer)
    {
        this.printer = printer;
    }

    public void doWriting(String str)
    {
        printer.print(str);
    }
}
```



```java
public class HP_Printer
{
    public void print(String str)
    {
        // print the string ..
    }
}
```

Good:

```
public interface ICanWrite
{
    public void write(String str);
}
```

```
public class HP_Printer implements ICanWrite
{
    @Override
    public void write(String str)
    {
        // print the string ..
    }

}
```

```
public class WritingManager
{
    ICanWrite writable;

    WritingManager(ICanWrite writable)
    {
        this.writable = writable;
    }

    public void doWriting(String str)
    {
        writable.write(str);
    }
}
```

```
public class Scodix_Printer implements ICanWrite
{
    @Override
    public void write(String str)
    {
        // print the string ..
    }

}
```

Good:

```java
public interface ICanWrite
{
     public void write(String str);
}
```

```java
public class WritingManager
{
    ICanWrite writable;

    WritingManager(ICanWrite writable)
    {
        this.writable = writable;
    }

    public void doWriting(String str)
    {
        writable.write(str);
    }
}
```

```java
public class HP_Printer implements ICanWrite
{
    @Override
    public void write(String str)
    {
        // print the string ..
    }

}
```

```java
public class Scodix_Printer implements ICanWrite
{
    @Override
    public void write(String str)
    {
        // print the string ..
    }

}
```

```java
public class FileWriter implements ICanWrite
{
    private String filePath;

    FileWriter(String filePaht)
    {
        this.filePath = filePath;
    }

    @Override
    public void write(String str)
    {
        // print the string to the file path
    }

}
```

The end