Software Architecture
Homework 1

Krikun Georgy

February 12, 2016

1 Question

a - Correct

```
public abstract class Square implements Shape {
  public abstract void setColor(String s);
}
```

In this case defined public abstract class with public abstract method setColor(String s) This method defined without body, and don't violate syntax.

b - Abstract methods do not specify a body

```
public abstract class Square implements Shape {
  public abstract void setColor(String s) {}
}
```

This variant trying define abstract method with empty body. But for abstract methods we can't define body, ever.

c - The type Shape cannot be the superclass of Square; a superclass must be a class

```
public abstract class Square extends Shape {
  public void setColor(String s) {}
}
```

For inheritance in Java we can extends some classes, but not interfaces. We can implement interfaces or extend interfaces by another, but can't extend class with interface.

d - Correct

```
public abstract class Square implements Shape {
  public void setColor(Integer i) {}
}
```

In this implementation we have correct definition of class Square, which implements interface Shape.

e - Correct

```
public abstract class Square implements Shape {
  public void setColor(String s) {}
  public void setColor(Integer i) {}
}
```

For overloading methods in Java we can define different methods with same name, for different types and arity of parameters

2 Question

a - correct

```
public Flower getType() { return this; }
```

Redefine method from Plant class, which return instance of Plant type but with new one, which return instance of Flower

b - incorrect

```
public String getType() { return "this"; }
```

We can't override method, what returns incompatible with superclass type

c - correct

```
public Plant getType() { return this;}
```

This method return Plant instance from Flower, downcast instance to Plant

d - correct

```
public Tulip getType() { return new Tulip(); }
```

As soon as Tulip inherited from Plant

e - incorrect

```
public String getType() { return this; }
```

This variant shows two violations:

- First one - type String not compatible with Plant type
- Second one - `this` - instance of Flower type, but method declared with String return type

3 Question

returns 9

On first step created static field `y`, of `Uber` class, and initialize with value 2. Then try create `Minor` instance, by calling constructor `Uber(int x)`. This constructor call default constructor for `Uber()`, which multiply value `y` twice. After this evaluation returns to `Minor()` constructor, which increment value `y` by three. So for static field y we have value 9, and as soon as value public we can access it from other classes.

4 Question

a - incorrect

```
PitBull p2 = (PitBull) dog1;
```

`dog1` created by effective class `Dog`, we can't cast parent class to child, because child class can be "bigger" than parent.

b - correct

```
PitBull p2 = (PitBull) dog2;
```

In this case `dog2` instance created by `PitBull` constructor, so in memory we have full `PitBull` instance, which downcast to `Dog` in under the hod, when we assign to `dog2`. So when we explicitly cast instance to `PitBull`, we can do it, because we have `PitBull` instance in memory.

c - incorrect

```
PitBull p2 = dog2;
```

But in this case, we didn't cast it explicitly. Compiler don't know, how to cast that, because in general we can't say, that instance on this reference is `PitBull`.

5 Question

a - incorrect

```
x2.do2();
```

The method `do2()` is undefined for the type `X` `x2` is instance of `X` class, so when we call `Y` constructor, we create `Y` instance, but cast it to `X` class (we can do it because `Y` extends `X`) But `X` class haven't got `do2` method.

b - correct

```
((Y)x2).do2();
```

This is legal call of `do2` method, `Y` class, because we create `x2` object with `Y` constructor.

c - incorrect

```
(Y)x2.do2();
```

Cast priority less than methods call. So we can't call method before cast to `Y` type. The method `do2()` is undefined for the type `X`.