Class Design

Constructor

cyclic reference

```java
class Person {
  Person(){
    this(4);
  }
  //does not compile! cyclic reference
  Person(int x){
    this();
  }
}
```

Abstract

Only instance methods can be marked abstract within a class!
Not variables, constructors, or static methods.

```java
public abstract class AnAbstractClass {
  abstract void hello();
  //Illegal combination of modifiers: 'abstract' and 'private'
  private abstract void notAllowed();  //does not compile!

  //Illegal combination of modifiers: 'final' and 'abstract'
  abstract final void notAllowed(); //does not compile!

  //Illegal combination of modifiers: 'abstract' and 'static'
  abstract static void notAllowed(); //does not compile!

  //Modifier 'abstract' not allowed here
  abstract AnAbstractClass(){} //does not compile!
}
```

abstract & final

abstract and **final** together, does not compile!

```
public abstract final class Tortoise { // DOES NOT COMPILE
    public abstract final void walk(); // DOES NOT COMPILE
}
```

abstract and private Modifiers

A method cannot be marked as both abstract and private.

```
public abstract class Whale {
    private abstract void sing(); // DOES NOT COMPILE
}
```

abstract & static

A method cannot be marked as both abstract and static.

```
abstract class Dog {
   abstract static void counter(); // DOES NOT COMPILE
}
```

Creating Immutable Objects

Declaring an Immutable Class

1. Mark the class as final or make all the constructors private.
2. Mark all the instance variables private and final.
3. Don’t define any setter methods.
4. Don’t allow referenced mutable objects to be modified.
5. Use a constructor to set all properties of the object, making a copy if needed.

Inheritance

Extends & Implements

Mind that extends come before implements!

```
class Jaguar extends Feline implements Run {...}
```

Override instance vs static methods

Instance method cannot override static method

Extends with static methods

Override/Overload private methods - wrong

There is no override/overload when the method in the parent class is marked private.

```
//no point to mark a method private & final as private methods cann
  private final void sayHello(){
        System.out.println("hello");
    }
```

Constructors

Constructors cannot be declared final

```
public class Man {
  public final Man() {} //DOES NOT COMPILE!
}
```

Overriding a method

Covariant return types

Specifically, when talking about return types in methods, covariant types mean that a subclass can have a more specific (i.e., a subtype) return type compared to its parent class.

```
public class Animal {
   protected CharSequence getName() {
      return "animal";
   }
   protected String getColor() {
      return "white";
   } }

public class Pet extends Animal {
   public String getName() {
      return "I am a kitten";
   }
   public CharSequence getColor() { // DOES NOT COMPILE
      return "red";
   } }
```

private final methods

Declaring a method as both private and final is redundant and has no additional impact on the method's behavior.

```
    private final void sayHello() {
      System.out.println("hello");
    }
```

Override private methods

```
  public class Insect {
     private String getSize() {
        return "Undefined";
     } }

  //this is not override, they are treated like two independent metho
  public class Fly extends Insect {
     private int getSize() {
        return 5;
     } }
```

About Override instance variables

Instance variables **are not overridden**, they are hidden. Polymorphism and overriding do not apply to instance variables.

Hiding Variables

Hiding Variables - 2

Parent - Child example

static final

```
  class Parent {}
      public static final void message() {
      }
  }
  class Child extends Parent {
    static void message() {  //DOES NOT COMPILE!!
      System.out.println("ciao");
    }
  }
```

hinding static methods The static method message() is marked final, so it cannot be hidden in the subclass.