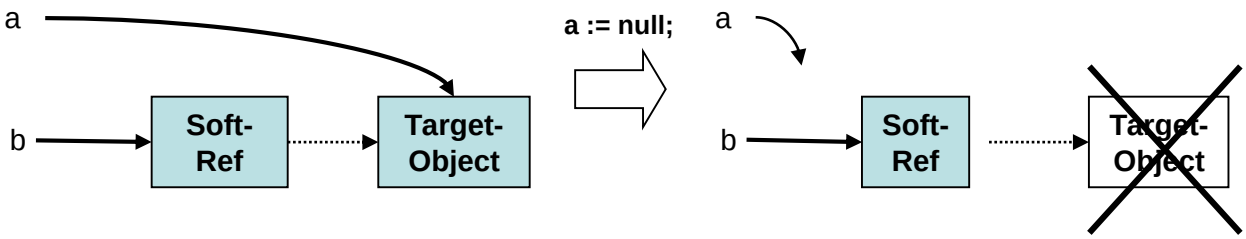
Worksheet Singleton: Weak References

Consider the following implementation of the singleton pattern:

```
public final class Singleton {
    private Singleton() { }
    private static Singleton instance = null;
    public static synchronized Singleton getInstance() {
        if(instance == null) instance = new Singleton();
        return instance;
    }
}
```

The singleton instance, which is referenced by the field instance declared in the Singleton class once created can never be freed by the garbage collector. If a singleton is no longer referenced by any other object, it could theoretically be collected by the GC in order to provide more memory to the rest of the application.

A soft reference (`java.lang.ref.SoftReference`) is an object that contains a special reference to another object. Specifically to this reference is that the referenced object can be deleted by the GC as soon as no strong (normal) references refer to this object.



Imagine a soft reference like a weak bridge that can break when the GC is trying to drive over it. Normal references (also called strong references) are, on the other hand, resistant to any traversing.

Task:
How does the implementation of the Singleton class look like if soft references are used?

Worksheet Singleton: Serialization

If a Singleton class is declared serializable, then a singleton instance can easily be copied. It must only be stored and then read again:

```
Singleton s1 = Singleton.getInstance();

FileOutputStream fos = new FileOutputStream("test.ser");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(s1);
oos.close();

FileInputStream fis = new FileInputStream("test.ser");
ObjectInputStream ois = new ObjectInputStream(fis);
Singleton s2 = (Singleton)ois.readObject();
ois.close();
```

Task:
Explain how you could put a stop to this trick using method `readResolve`.

References:

- Java Object Serialization Specification , Kapitel 3.7
  https://docs.oracle.com/javase/9/docs/specs/serialization/input.html#the-readresolve-method
- J. Bloch, Effective Java, Item 3,
  http://dl.softgozar.com/Files/Ebook/Effective_Java_Second_Edition_Softgozar.com.pdf
- R.J. Lorimer, Serialization: Understand 'readResolve', Javalobby,
  http://www.javalobby.org/java/forums/t17491.html

Worksheet Singleton: Initialization on Demand Holder Idiom

The following Singleton implementation is known as *Initialization-on-demand holder idiom*.

```
public final class Singleton {
    private static class Holder {
        private static final Singleton INSTANCE = new Singleton();
    }

    private Singleton() {}

    public static Singleton getInstance(){
        return Holder.INSTANCE;
    }
}
```

Question:

What is the advantage of the above implementation over the following implementation that we have seen in the lecture?

```
public final class Singleton {
    private Singleton() { }
    private static Singleton instance = new Singleton();
    public  static getInstance(){ return instance; }
}
```

And also over the following implementation?

```
public final class Singleton {
    private Singleton() { }
    private static Singleton instance = null;
    public synchronized static getInstance(){
        if (instance == null) instance = new Singleton();
        return instance;
    }
}
```

Reference:
http://en.wikipedia.org/wiki/Initialization_on_demand_holder_idiom