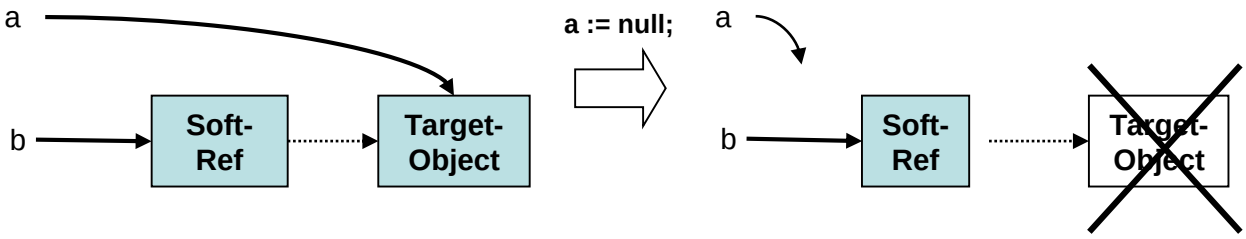
Arbeitsblatt Singleton: Schwache Referenzen

Betrachten wir folgende Implementierung des Singleton Patterns:

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

Die Instanz, die in der Klasse Singleton über das Feld `instance` referenziert wird, kann vom Garbage Collector nie mehr weggeräumt werden. Wird ein Singleton jedoch von keiner anderen Instanz mehr referenziert, so könnte es vom GC eigentlich eingesammelt werden, um dem Rest der Applikation mehr Speicher zur Verfügung zu stellen.

Eine Soft-Reference (`java.lang.ref.SoftReference`) ist ein Objekt, das eine spezielle Referenz auf ein anderes Objekt enthält. Speziell an dieser Referenz ist, dass das referenzierte Objekt vom GC gelöscht werden kann, sobald keine normalen Referenzen mehr darauf zeigen.



Stellen Sie sich eine Soft-Reference wie eine schwache Brücke vor, die einbrechen kann, wenn der GC versucht darüber zu fahren. Normale Referenzen (auch *strong references* genannt) halten hingegen jeder Traversierung stand.

Aufgabe:
Wie sieht die Implementierung der Singleton-Klasse mit Soft-Referenzen aus?

Arbeitsblatt Singleton: Serialisierung

Falls eine Singleton-Klasse als serialisierbar deklariert ist, kann eine Singleton-Instanz leicht kopiert werden. Sie muss dazu nur gespeichert und dann wieder eingelesen werden:

```
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();
```

Aufgabe:
Erklären Sie, wie dies mit Hilfe der Methode `readResolve` verhindet werden kann.

Referenzen:

- 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

Arbeitsblatt Singleton: Initialization on Demand Holder Idiom

Folgende Singleton-Implementierung ist unter dem Namen *Initialization-on-demand holder idiom* bekannt.

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

    private Singleton() {}

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

Frage:

Was ist der Vorteil dieser Implementierung gegenüber der Implementierung die wir in der Vorlesung gesehen haben:

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

und auch gegenüber der folgenden Implementierung:

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

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