Exercise: Singleton

- In some situations, you may create the only instance of the class.

```java
public class Singleton {

    // Do now allow to invoke the constructor by other classes.
    private Singleton() {}

    // Will be ready as soon as the class is loaded.
    private static Singleton instance = new Singleton();

    // Only way to obtain this singleton by the outside world.
    public static Singleton getInstance() {
        return instance;
    }
}
```
Object Elimination: Garbage Collection (GC)

- Java handles object deallocation by GC.
  - Timing: preset period or when memory stress occurs.
- GC is a daemon thread, which searches for those unreferenced objects.
  - An object is unreferenced when it is no longer referenced by any part of your program. (How?)
  - To make the object unreferenced, simply assign null to the reference.
- Note that you may invoke `System.gc()` to execute the deallocation procedure.
  - However, frequent invocation of GC is time-consuming.

---

HAS-A Relationship

- **Association** is a weak relationship where all objects have their own lifetime and there is no ownership.
  - For example, teacher ↔ student; doctor ↔ patient.
- If A uses B, then it is an aggregation, stating that B exists independently from A.
  - For example, knight ↔ sword; company ↔ employee.
- If A owns B, then it is a composition, meaning that B has no meaning or purpose in the system without A.
  - For example, house ↔ room.
Example: Lines (Aggregation)

- +2: two **Point** objects used in one **Line** object.
public class Line {
    private Point head, tail;

    public Line(Point p1, Point p2) {
        head = p1;
        tail = p2;
    }

    /* ignore some methods */

    public double length() {
        return head.getDistanceFrom(tail);
    }
}

More Examples

- Circle, Triangle, and Polygon.
- Book with Authors.
- Lecturer and Students in the classroom.
- Zoo with many creatures, say Dog, Cat, and Bird.
- Channels played on TV.
- More.
First IS-A Relationship: Class Inheritance

- We can define new classes by inheriting states and behaviors commonly used in predefined classes (aka prototypes).
- A class is a subclass of some class, which is so-called the superclass, by using the extends keyword.

For example,

```java
// superclass
class A {
    void doAction() {} // A can run doAction().
}

// subclass
class B extends A {} // B can also run doAction().
```

- Note that Java allows single inheritance only.
Constructor Chaining

- Once the constructor is invoked, JVM will invoke the constructor of its superclass (recursively).
- You might think that there will be a whole chain of constructors called, all the way back to the constructor of the class `Object`, the topmost class in Java.
- In this sense, we could say that every class is an immediate or a distant subclass of `Object`. 
Illustration for Class Hierarchy

See Fig. 3-1 in p. 113 of Evans and Flanagan.

Zheng-Liang Lu
Java Programming
• Recall that `this` is used to refer to the object itself.
• You can use `super` to refer to (non-private) members of the superclass.
• Note that `super()` can be used to invoke the constructor of its superclass, just similar to `this()`.
Method Overriding (1/2)

- A subclass is supposed to **re-implement** the methods inherited from its superclass.
- For example, `toString()` is inherited from `Object`.
  - This method will be invoked by `println()`.
  - It returns the hashcode\(^3\) of the object by default.
  - It could be **overridden** so it returns a string of desirable information.

\(^3\)See https://en.wikipedia.org/wiki/Java_hashCode().
Method Overriding (2/2)

• The requirement of method overriding is as follows:
  • Method signature identical to the one of its superclass;
  • Same return type;
  • Non-reduced visibility relative to the one of its superclass.\(^4\)
• Note that you cannot override the static methods.
• You could invoke the overridden method by using `super`.
• You should use the annotation\(^5\) `@Override` to help you.

```java
class B extends A {
    @Override
    void doAction() { /* new impl. w/o changing API */ }
}
```

---

\(^4\)For example, you cannot reduce the visibility from public to private.

\(^5\)See https://docs.oracle.com/javase/tutorial/java/annotations/.
Subtype Polymorphism

- The word polymorphism literally means “many forms.”
- Subtype polymorphism allows you to create a single interface to different implementations.
- How to make a “single” interface for different implementations?
  - Use the superclass of those types as the placeholder.
  - Key idea: Program to abstraction, not to implementation.

Example: Dependency Reduction (Decoupling)

```java
class Student {
    void doMyJob() { /* Do not know the detail yet. */ }
}

class HighSchoolStudent extends Student {
    void doHomework() {}  
    @Override
    void doMyJob() { doHomework(); }
}

class CollegeStudent extends Student {
    void writeFinalReports() {}  
    @Override
    void doMyJob() { writeFinalReports(); }
}
```
public class PolymorphismDemo {

    public static void main(String[] args) {
        HighSchoolStudent h = new HighSchoolStudent();
        goStudy(h);
        CollegeStudent c = new CollegeStudent();
        goStudy(c);
    }

    public static void goStudy(Student s) {
        s.doMyJob();
    }

    /* no need to write these methods
    public static void goStudy(HighSchoolStudent s) {
        s.doHomework();
    }

    public static void goStudy(CollegeStudent s) {
        s.writeFinalReports();
    }
    */
}