Assertions

• Assertions are checks for internal consistency, to assure program correctness. They are to reaffirm your assumption

• They are useful for the programmer when writing and debugging. Once done, you can turn off assertion (actually they are off by default, you have to turn them on for testing) so that your program runs faster. Run with argument `java -ea filename` for `enable-assertion`

• Do not use assertion instead of exceptions - which are supposed to stay on during program execution as a way to handle unusual circumstances and interact with calling methods.

```java
public void withdraw(double amount) throws OverdrawnException {
    if (amount > balance){
        throw new OverdrawnException("trying to withdraw " + amount + " but balance is " + balance);
    }
    else
        balance -= amount;

    assert balance>=0;
}
```
Abstract classes and methods

- an abstract method provides a `design spec' but not a realization of a method. A subclass that extends the class must provide the concrete implementation to make it usable.

- an abstract method means the class itself becomes abstract. (cannot create an instance using new, but concrete subclasses will still invoke the superclass constructor)

- but abstract method allows for polymorphism: use the superclass to hold subclass objects and invoke the appropriate methods (as on p. 499 example in Liang text)

- UML: italics for abstract. # for protected variables (recall: - for private, + for public)

```java
public abstract class GymMachine {
    . . .

    public abstract method show(){
        // implement method that
        // display machine on canvas
    }
}
```

each subclass, Treadmill, Stepper, and Bike, will have their own show() method
Interfaces

- like a class, contains only constants and abstract methods

- used to specify certain kind of behavior, across different classes. Specify an interface, and other class implement the interface

- like inheritance, is a way to get around java’s single inheritance

- can declare variables to be of the same type as the interface

- an important one is the Comparable interface

```java
public interface Comparable<E> {
    public int compareTo(E o);
}
```

```java
public class MyTime implements Comparable<MyTime> {
    @Override
    public int compareTo(MyTime obj){
        // return neg,0 or pos if `this' is <=,=,>= obj
    }
}
```

- this allows writing methods e.g. that sort items of any type, as long as that class that implements Comparable