ASSERTIONS

Assertions

- An assertion statement allows you to test an assumption
  - You assert what you assume to be true
- Syntax:

```
assert boolean;

assert boolean : expression; // expression is a message
```

- When an assert fails, it throws an AssertionError.
- Assertions are disabled by default, but you can enable them with the –ea flag.

Assertions

- Assertions can help in development, testing, and debugging.
- Assertions can also help document your program, supporting maintenance.

Assertions are **NOT**…

- A replacement for conditionals
  - Use conditionals to control the logical flow of your program
- A replacement for exception handling
  - Use exception handling to account for known potential error situations

- Do **not** use assertions to check parameters of public methods.
- Do **not** use assertions to do any work that is needed when your program is live and running.

Assertions are good for…

- Invariants
  - internal, control-flow, class
- Preconditions
- Postconditions

Internal Invariants

- Something that should *always* be true

```
if(i%3==0) {
    …
} else if(i%3==1) {
    …
} else {
    …
}
```

Internal Invariants

```
if(i%3==0) {
    …
} else if(i%3==1) {
    …
} else { // i%3==2
    …
}
```

Internal Invariants

```
if(i%3==0) {
    …
} else if(i%3==1) {
    …
} else {
    assert (i%3==2);
    …
}
```

Control-Flow Invariants

- Code that should *never* be reached

```
switch(semester) {
    case(Semester.FALL):
        …
        break;
    case(Semester.SPRING):
        …
        break;
    case(Semester.SUMMER):
        …
        break;
    default:
        assert false;
}
```

Control-Flow Invariants

```
switch(semester) {
    case(Semester.FALL):
        …
        break;
    case(Semester.SPRING):
        …
        break;
    case(Semester.SUMMER):
        …
        break;
    default:
        throw new AssertionError(semester);
}
```

Control-Flow Invariants

- Can use `assert false;` anywhere that should never be reached

```
public int method() {
     for(loop where you expect to find the answer) {
          if(you find the answer)
               return the answer'
     }
     assert false;
     return -1;
}
```

Control-Flow Invariants

- Can alternatively throw an AssertionError

```
public int method() {
     for(loop where you expect to find the answer) {
          if(you find the answer)
               return the answer'
     }
     throw new AssertionError();
     // note you now don't need a return!
}
```

Class Invariants

- Something that should always be true about the state of an object
- Example:
  - Create a private hasValidState() method that checks the condition of the object's structure
  - Include an `assert hasValidState();` at the end of any method that changes the state of the object

Class Invariants

- Example:

```
private int[] nums = …

public boolean arrayFilledPos() {
     for(int i=0; i<nums.length; i++)
      if(nums[i]<=0)
           return false;
     return true;
}

assert arrayFilledPos();
// put this at the end of methods that manipulate the array
```

Conditions

- Preconditions are what you expect/demand to be true when you begin a task (method)
  - Characteristics of the inputs
- Postconditions are what you promise to be true when you end a task (method)
  - Characteristics of the outputs

Preconditions

- For preconditions on public methods, use explicit checks **not** assertions.
  - An assert won't work unless it's enabled
  - An assert would throw the wrong kind of exception
- Do not use assertions to check preconditions of public methods!
- You **can** use assertions to check preconditions for non-public methods.
- Assertions can be used to confirm that a lock is held within a private method.

Preconditions- Example

```
public void deposit(double depositAmount) {
      // use a conditional, not an assertion
      if(depositAmount <= 0) {
            throw new IllegalArgumentException
                  (“Deposits must be positive”);
      }
      makeDeposit(depositAmount);
}
private void makeDeposit(double amt) {
      assert amt>0;
      balance += amt;
}
```

Preconditions- Locks Example

```
public synchronized void deposit(double depositAmount) {
      if(depositAmount <= 0) {
            throw new IllegalArgumentException
                  (“Deposits must be positive”);
      }
      makeDeposit(depositAmount);
}
private void makeDeposit(double amt) {
      assert amt>0;
      assert Thread.holdsLock(this);
      balance += amt;
}
```

Postconditions

- You can use assertions at the end of public and private methods.

Postconditions

```
public void deposit(double depositAmount) {
    if(depositAmount <= 0) {
        throw new IllegalArgumentException
            (“Deposits must be positive”);
    }
    double oldBalance = balance;
    makeDeposit();
    assert balance > oldBalance;
}
```

Summary

- Use conditionals for checking conditions that are part of the logic of the program.
  - Conditionals can be used to validate input of public methods.
- Use exceptions to handle errors that might occur during the execution of the program.
  - Exceptions can be used to ensure proper execution of a program.
  - Exceptions can be used to validate input of public methods.
- Use assertions to check for errors that should never happen.
  - Use assertions to state things you already known to be true.
  - Use assertions during testing and debugging.
  - Do not rely on assertions to ensure proper execution of a program.