Assertions, pre/post-conditions

Assertions: Section 4.2 in Savitch (p. 239)

Programming as a contract

- Specifying what each method does
  - Specify it in a comment before method's header
- Precondition
  - What is assumed to be true before the method is executed
  - **Caller obligation**
- Postcondition
  - Specifies what will happen if the preconditions are met – what the method guarantees to the caller
  - **Method obligation**

Example

```
/*
 ** precondition:  x >= 0
 ** postcondition: return value satisfies:
 ** result * result == x
*/
double sqrt(double x) {

}
```

Enforcing preconditions

```
/*
 ** precondition:  x >= 0
 ** postcondition: return value satisfies:
 ** result * result == x
*/
double sqrt(double x) {
  if (x < 0)
     throw new ArithmeticExpression ("you
       tried to take sqrt of a neg number!");

}
```

What is an assertion?

- An *assertion* is a statement that says something about the state of your program
- Should be true if there are no mistakes in the program

```
//n == 1
while (n < limit) {
  n = 2 * n;
}
// what will be the state here?
```

What is an assertion?

- An *assertion* is a statement that says something about the state of your program
- Should be true if there are no mistakes in the program

```
//n == 1
while (n < limit) {
  n = 2 * n;
}
//n >= limit
```

assert

Using **assert:**

```
assert n == 1;
while (n < limit) {
   n = 2 * n;
}
assert n >= limit;
```

When to use Assertions

- Another example

```
if (i % 3 == 0) { ... }
else if (i % 3 == 1) { ... }
else { // We know (i % 3 == 2)
... }
```

When to use Assertions

- We can use assertions to guarantee the behavior.

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

Another example

```
int p=…,d=…;
int q = p/d;
int r = p%d;
assert ?
```

Another example

```
int p=…,d=…;
int q = p/d;
int r = p%d;
assert p == q*d + r;
```

Performance

- Assertions may slow down execution. For example, if an assertion checks to see if the element to be returned is the smallest element in the list, then the assertion would have to do the same amount of work that the method would have to do
- Therefore assertions can be **enabled** and **disabled**
- Assertions are, by default, disabled at run-time
- In this case, the assertion has the same semantics as an empty statement
- Think of assertions as a debugging tool
- Don't use assertions to flag user errors, because assertions can be turned off

Assertions in Eclipse

- To enable assert statements, you must set a compiler flag. Go to Run -> Run Configurations -> Arguments, and in the box labeled VM arguments, enter either -enableassertions or just -ea

Control Flow

- If a program should never reach a point, then a constant false assertion may be used

```
private void search() {
    for (...) {
        ...
        if (found) // will always happen
            return;
    }
    assert false; // should never get here
}
```

Assertions

- Syntax:
  `assert Boolean_Expression;`
- Each assertion is a Boolean expression that you claim is true.
- By verifying that the Boolean expression is indeed true, the assertion confirms your claims about the behavior of your program, increasing your confidence that the program is free of errors.
- If assertion is false when checked, the program raises an exception.

Assertions in switch statements

```
switch(suit) {
  case Suit.CLUBS:
    ...
  break;
  case Suit.DIAMONDS:
    ...
  break;
  case Suit.HEARTS:
    ...
    break;
  case Suit.SPADES:
      ...
}
```

If your program is correct, one of these cases should hold!

How to use assertions to verify that?

Assertions in switch statements

```
switch(suit) {
  case Suit.CLUBS:
    ...
  break;
  case Suit.DIAMONDS:
    ...
  break;
  case Suit.HEARTS:
    ...
    break;
  case Suit.SPADES:
      ...

  default:
    assert false;
}
```

Assertions in switch statements

```
switch(suit) {
  case Suit.CLUBS:
    ...
  break;
  case Suit.DIAMONDS:
    ...
  break;
  case Suit.HEARTS:
    ...
    break;
  case Suit.SPADES:
      ...

  default:
    assert false : suit;
    //gives the value that violated the assertion
}
```

Assertions

Let's take a closer look at the assertion statement:

```
assert false : suit;
```

This uses a more general form of the assert statement:

assert *Expression*$_1$: *Expression*$_2$;

- *Expression*$_1$ is a boolean expression.
- *Expression*$_2$ is an expression that has a value. (It cannot be an invocation of a method that is declared void.)
- Use this version of the assert statement to provide a message with the AssertionError. The system passes the value of *Expression*$_2$ to the appropriate AssertionError constructor

Assertions in switch statements

```
switch(suit) {
  case Suit.CLUBS:
    ...
  break;
  case Suit.DIAMONDS:
    ...
  break;
  case Suit.HEARTS:
    ...
    break;
  case Suit.SPADES:
      ...

  default:
    throw new AssertionError(suit);  //an acceptable alternative
}
```

When to use assertions?

- Programming by contract
- **Preconditions** in methods (eg value ranges of parameters) should be enforced rather than asserted because assertions can be turned off
- **Postconditions**
  - Assert post-condition

Class Invariants

- A **class invariant** is a condition that all objects of that class must satisfy while it can be observed by clients
- Example: your bank balance should always be positive
- Verify a class invariant using assertions