Special Topic 8.2

Class Invariants

Special Topic 6.5 introduced the concept of **loop invariants**. A loop invariant is established when the loop is first entered, and it is preserved by all loop iterations. We then know that the loop invariant must be true when the loop exits, and we can use that information to reason about the correctness of a loop.

Class invariants fulfill a similar purpose. A class invariant is a statement about an object that is true after every constructor and that is preserved by every mutator (provided that the caller respects all preconditions). We then know that the class invariant must always be true, and we can use that information to reason about the correctness of our program.

Here is a simple example. Consider a `BankAccount` class with the following preconditions for the constructor and the mutators:

```
public class BankAccount
{
   . . .
   /**
      Constructs a bank account with a given balance.
      @param initialBalance the initial balance
      (Precondition: initialBalance >= 0)
   */
   public BankAccount(double initialBalance) { . . . }
   {
      balance = initialBalance;
```

```
    }

    /**
       Deposits money into the bank account.
       @param amount the amount to deposit
       (Precondition: amount >= 0)
    */
    public void deposit(double amount) { . . . }

    /**
       Withdraws money from the bank account.
       @param amount the amount to withdraw
       (Precondition: amount <= getBalance())
    */
    public void withdraw(double amount) { . . . }
}
```

Now we can formulate the following class invariant:

```
getBalance() >= 0
```

To see why this invariant is true, first check the constructor; because the precondition of the constructor is

```
initialBalance >= 0
```

we can prove that the invariant is true after the constructor has set `balance` to `initialBalance`.

Next, check the mutators. The precondition of the `deposit` method is

```
amount >= 0
```

We can assume that the invariant condition holds before calling the method. Thus, we know that `balance >= 0` before the method executes. The laws of mathematics tell us that the sum of two nonnegative numbers is again nonnegative, so we can conclude that `balance >= 0` after the completion of the `deposit`. Thus, the `deposit` method preserves the invariant.

A similar argument shows that the `withdraw` method preserves the invariant.

Because the invariant is a property of the class, you document it with the class description:

```
/**
   A bank account has a balance that can be changed by
   deposits and withdrawals.
   (Invariant: getBalance() >= 0)
*/
public class BankAccount
{
   . . .
}
```