Task 3: Specify and verify the correctness of an interface implementation (with JML)

In our final task, we will specify an interface and verify that an implementation of that interface conforms to the specification. This will probably be the most interesting application for java developers, because it allows us to actually enforce the conditions the kind of properties that can only be informally suggested without JML. For example, if we create an interface called `Circle` which contains methods called `getCircumference` and `getRadius` with the obvious signatures, there is nothing to stop someone writing a class to implement that interface which overrides `getCircumference` with some number *other* than 2πr. The interfaces can only constrain the signatures of the methods that their implementations override, they cannot constrain their behaviour. JML lets get around this limitation by allowing us to verify that an implementation of an interface conforms to the JML specification of that interface, which is what we will do in this task.

We will return to our code from task 1, this time introducing an interface. Since much of the code will be re-usable from task 1, this task will go much more quickly.

1. Create another BankAccount class

```
public class BankAccount implements IBankAccount {
    private int balance;


    public BankAccount(int initialBalance) {
        balance = initialBalance;
    }

    public void withdraw(int amount) {
        if ((long) balance - (long) amount < Integer.MIN_VALUE) {
        throw new IllegalArgumentException("Withdrawal would cause
underflow");
    }
    balance -= amount;
}

    public int getBalance() {
        return balance;
    }
}
```

PROF

2. Create an BankAccount interface

Write a minimal interface for our BankAccount class to implement.

```
public interface IBankAccount {
    void withdraw(int amount);
    int getBalance();
}
```

3. Create specification for getBalance interface method

Use the same specification for `getBalance` as in task 1.

```
public interface IBankAccount {
   //@ ensures \result >= Integer.MIN_VALUE;
   /*@ pure @*/ int getBalance();

   void withdraw(int amount);
}
```

4. Create specification for withdraw interface method

User the same interface for `withdraw` as in task 1.

```
public interface IBankAccount {
   //@ ensures \result >= Integer.MIN_VALUE;
   /*@ pure @*/ int getBalance();

   //@ requires amount >= 0 && ((long)getBalance() - (long)amount) >=
Integer.MIN_VALUE;
   //@ ensures getBalance() == \old(getBalance()) - amount;
   //@ also
   //@ public exceptional_behavior
   //@ requires amount >= 0 && ((long)getBalance() - (long)amount) <
Integer.MIN_VALUE;
   //@ signals_only IllegalArgumentException;
   void withdraw(int amount);
}
```

PROF

5. Make balance spec_public

```
public class BankAccount implements IBankAccount {
    private /*@ spec_public @*/ int balance;
```

```
    public BankAccount(int initialBalance) {
        balance = initialBalance;
    }

    public void withdraw(int amount) {
        if ((long) balance - (long) amount < Integer.MIN_VALUE) {
        throw new IllegalArgumentException("Withdrawal would cause
underflow");
    }
    balance -= amount;
}

    public int getBalance() {
        return balance;
    }
}
```

6. Make instance variables assignable

```
public class BankAccount implements IBankAccount {
    private /*@ spec_public @*/ int balance;
    //@ assignable \everything;

    public BankAccount(int initialBalance) {
        balance = initialBalance;
    }

    public void withdraw(int amount) {
        if ((long) balance - (long) amount < Integer.MIN_VALUE) {
        throw new IllegalArgumentException("Withdrawal would cause
underflow");
    }
    balance -= amount;
}

    public int getBalance() {
        return balance;
    }
}
```

PROF

7. Write specification for BankAccount class constructor

```
public class BankAccount implements IBankAccount {
    private /*@ spec_public @*/ int balance;
    //@ assignable \everything;

    //@ ensures balance == initialBalance;
    public BankAccount(int initialBalance) {
```

```
        balance = initialBalance;
    }

    public void withdraw(int amount) {
        if ((long) balance - (long) amount < Integer.MIN_VALUE) {
        throw new IllegalArgumentException("Withdrawal would cause
underflow");
    }
    balance -= amount;
}

    public int getBalance() {
        return balance;
    }
}
```

8. Write specification for BankAccount class withdraw method

```
public class BankAccount implements IBankAccount {
    private /*@ spec_public @*/ int balance;
    //@ assignable \everything;

    //@ ensures balance == initialBalance;
    public BankAccount(int initialBalance) {
        balance = initialBalance;
    }
    //@ also
    //@ requires amount >= 0 && ((long)getBalance() - (long)amount) >=
Integer.MIN_VALUE;
    //@ ensures getBalance() == \old(getBalance()) - amount;
    //@ also
    //@ public exceptional_behavior
    //@ requires amount >= 0 && ((long)getBalance() - (long)amount) <
Integer.MIN_VALUE;
    //@ signals_only IllegalArgumentException;
    public void withdraw(int amount) {
        if ((long) balance - (long) amount < Integer.MIN_VALUE) {
        throw new IllegalArgumentException("Withdrawal would cause
underflow");
    }
    balance -= amount;
}


    public int getBalance() {
        return balance;
    }
}
```

PROF

9. Write specification for BankAccount class getBalance method

```
public class BankAccount implements IBankAccount {
    private /*@ spec_public @*/ int balance;
    //@ assignable \everything;

    //@ ensures balance == initialBalance;
    public BankAccount(int initialBalance) {
        balance = initialBalance;
    }
    //@ also
    //@ requires amount >= 0 && ((long)getBalance() - (long)amount) >=
Integer.MIN_VALUE;
    //@ ensures getBalance() == \old(getBalance()) - amount;
    //@ also
    //@ public exceptional_behavior
    //@ requires amount >= 0 && ((long)getBalance() - (long)amount) <
Integer.MIN_VALUE;
    //@ signals_only IllegalArgumentException;
    public void withdraw(int amount) {
        if ((long) balance - (long) amount < Integer.MIN_VALUE) {
        throw new IllegalArgumentException("Withdrawal would cause
underflow");
    }
    balance -= amount;
}

    //@ also
    //@ ensures \result == balance;
    public /*@ pure @*/ int getBalance() {
        return balance;
    }

}
```

PROF

10. Verify specification

At this point we have annotated our interface and our class with the appropriate specifications, and we can proceed to verify that the class in fact conforms to the specification.

Command: `./openJML -esc BankAccount.java IBankAccount.java`