# CS-115 Lab Class 8 16/03/2020

## ☐ Task 8.1

In this lab we are going to play around with sets in Java. We are going to create sets that contain bank account objects. Remember from your mathematics studies, that sets contain unique elements (i.e., a set cannot contain two elements which are considered equal). The sets in java will not work (correctly) out of the box, we will have to implement some methods to get them to work.

### Part 1

Create a class `BankAccount` with

- account number (int), and
- sort code (int).

The constructor should take an account number and a sort code as arguments. Once an instance of BankAccount has been created it should be impossible to change either the account number or the sort code.

Create a main class and check your BankAccount class works as expected.

### Part 2

Java provides various set data types, each one using a different data structure to implement the operations of a set. We are going to use a `HashSet` to store `BankAccount` objects. A `HashSet` uses a `HashMap` to implement a set.

Write a main method which does exactly the following:

- Creates a `HashSet` that stored `BankAccount` objects.
- Adds the following separate instances of `BankAccount` to the hash set:
  - Bank Account 1 with account number 123000 and sort code 321000.
  - Bank Account 2 with account number 555000 and sort code 555000.
  - Bank Account 3 with account number 123000 and sort code 321000.
- Loops over the hash set using a for-each loop printing the account number and sort code of each bank account as it goes.

Run your code and notice how the output shows that the set contains 3 bank account instances. Now notice that the first and third objects should be counted as equal (as they have the same account number and sort code). As sets do not contain multiple occurrences of equal objects, we should only have 2 objects in the set – not 3! We need to fix this!

The `HashSet` uses both the `BankAccount`'s `equals` method and `hashCode` method (both inherited from the class `Object`). The problem is that the inherited versions are not good enough. They consider bank account 1 and bank account 3 to be unequal, where as we want them to be considered as equal. We need to override the method.

**Part 3**

Implement a better `equals` method for the class `BankAccount`. The method needs to be declared as

```
public boolean equals(Object obj)
```

This method should return `true` if the `this` reference and the `obj` reference each point to objects which are considered equal. It should return `false` if the objects are considered unequal.

The slight complication is the type of the `obj` reference: it is of type `Object`. This means we need to be able to check if a particular BankAccount is equal to, say, a particular `Eagle` or `SpaceShip` - which of course it would not be.

Use the following template as a starting point and implement an equals method for the class `BankAccount` that regards two `BankAccount` objects as equal if and only if they have the same account number and sort code.

```
public boolean equals(Object obj) {
    // If the references match then we then this and obj point to
    // the same object. Therefore, they are equal.
    if(this == obj) {
        return true;
    }

    // This object (which exists) can never be equal to a null
    // reference.
    if (obj == null) {
        return false;
    }

    // Now we know both objects exist. We need to check that their
    // classes match.
    if(obj.getClass() != this.getClass())
    {
        return false;
    }

    // Both this and obj exist and are of the same class.
    // Now we can compare the account numbers and sort codes.
    BankAccount other = (BankAccount) obj;
    < place your code here >
}
```

Test your code. In your main method (comment out the previous code and) call the equals method on two bank account objects which should be considered equal – they method should return true. Call the method on two unequal methods and check that the result is as you expect.

Once you have done this, then all that is left is the `hashCode` method.

**Part 4**

The BankAccount's hashCode method must be declared as:

```
public int hashCode()
```

This should return an integer which is the hash of the object. A hash is a long number which indicates if two objects are to be considered equal. Idea: Take two complex pieces of data (these might be objects, files, strings, binary strings, etc) run them through the hash function, if the produced hashes are exactly the same then the objects are equal.

**Important: If two instances are considered equal by the equals method then the hashCode method must return exactly the same hash for both instances.**

Implement the hashCode method for the BankAccount class. To do this, take each of your non-static attributes, multiply each of them by a unique prime number, and sum them.

Test your code. In your main method (comment out the previous code and) call the hashCode method on the various bank account objects. Check the hashes match for equal bank accounts (and don't match for unequal objects).

**Part 5**

Revisit Part 2. Run the code again. Now the HashSet should work correct and only store 2 bank accounts. The third bank account that we attempt to add is disregarded (not added) as it is considered equal to the first. Hooray - Working sets in Java!

Java's standard library comes with many data structures ready to use, but many of them need certain methods (such as equals and hashCode) or interfaces to be implemented correctly before you can use them.

## ☐ Challenge Task 8.2

We have used a HashSet in this lab. Try and create a set of BankAccount objects using a TreeSet.