EBU5304 Software Engineering: Design Patterns Exercise 2

Consider the following interface type:

```
interface Comparator<T>
{
 public int compare(T o1, T o2);
}
```

It is provided as part of the `java.util` package so can be accessed by heading your code with:

```
import java.util.*;
```

An object of type `Comparator<T>` can be used to sort a `List<T>` collection in the order defined by the comparator object. If `list` is of type `List<T>` and `comp` is of type `Comparator<T>` for any object type `T`, then:

```
Collections.sort(list,comp);
```

will rearrange the contents of the list so they are in the order given by the comparator object. This is a simple example of the Strategy Design Pattern: delegating a task to a separate object that is passed as an argument to the code.

Now write four classes that implement `Comparator<BankAccount>`:

1) One that compares `BankAccount` objects by their account number

2) One that compares `BankAccount` objects by their account name

3) One that compares `BankAccount` objects by their balance

4) One that compares `BankAccount` objects by the closeness of their balance to a value provided by the constructor of the `Comparator<BankAccount>` object. So if the constructor is given value `600`, and `acc1` refers to a `BankAccount` object with balance `550` and `acc2` refers to a `BankAccount` object with balance `700`, it will judge the object referred to by `acc1` less than the object referred to by `acc2` because its balance is closer to `600`.

Use an object of each of these classes to sort a list of `BankAccount` objects.