Defensive Programming

And here too we tried to make our application as robust as possible, by adding default values, handling exceptions appropriately and so on and so forth.

An example for the usage of default values

```
public static SalutationType getSalutationType(String string) {
    switch(string) {
        case "mr":
            return SalutationType.MR;
        case "mrs":
            return SalutationType.MRS;
        case "family":
            return SalutationType.FAMILY;
        case "empty":
            return SalutationType.EMPTY;
        default:
            return SalutationType.EMPTY;
    }
}
```

And once again we extensively made use of the Enumeration type as its principle of encapsulation is quite handy to say the least.

```
public enum CustomerType {
    CONSUMER(){

        @Override
        public String toString(){
            return "consumer";
        }
    },
    BUSINESS(){

        @Override
        public String toString(){
            return "business";
        }
    };


    public static CustomerType getCustomerType(String string) {
      switch(string) {
          case "business":
              return CustomerType.BUSINESS;
          case "consumer":
              return CustomerType.CONSUMER;
          default:
              return CustomerType.CONSUMER;
      }
    }
}
```

If there's one thing we've taken away from the first task it's that defensive programming can be a real beauty. Its usefulness is not really up for debate anyway.