Unit 1 - Extras

JAVA OOP reinforcement

Basic inheritance

```
public class Store {
      public void welcome() {
      System.out.println("Welcome to our store!");
      }
}
public class LiquorStore extends Store {
      @Override
      public void welcome() {
            super.welcome();
            System.out.println("If you are younger than 18, go back
            home!");
      }
}
// MAIN
LiquorStore lqStore = new LiquorStore();
lqStore.welcome();
```

```
“Welcome to our store!” → super method call
"If you are younger than 18, go back home!"
```

Jose Ramón García Sevilla *jrgarcia@iesmarenostrum.com*

Abstract classes

```
public abstract class Store {
      public abstract void welcome();
}
public class LiquorStore extends Store {
     @Override
     public void welcome() {
         System.out.println("Welcome to our liquor store. "
         + "If you are younger than 18, go back home!");
         }
}
// MAIN
Store store = new Store(); → ERROR
LiquorStore liqStore = new LiquorStore(); → OK
```

Interfaces

```
public interface Startable {
     public void start();
}
public interface Stoppable {
     public void stop();
}
public class Student implements Startable, Stoppable {
     @Override
     public void start() {
          System.out.println("Program started.");
     }
     @Override
     public void stop() {
          System.out.println("Program stopped.");
     }
}
```

Polymorphism

```
public abstract class Store {...}
public class LiquorStore extends Store {
...
  public void buyLiquor() {
    System.out.println("Do you want beer, wine, rum, whisky
    or vodka?");
  }
}
// MAIN
Store store = new LiquorStore(); // this LiquorStore behaves
as a generic Store
store.buyLiquor(); // ERROR. We only can access Store methods
store.welcome(); // Ok, and executes LiquorStore
implementation
```

Polymorphism (II)

```
if( store instanceof LiquorStore ) {
    LiquorStore liqStore = (LiquorStore) store;
    liqStore.buyLiquor(); // OK
}

OR

if( store instanceof LiquorStore ) {
    ((LiquorStore) store).buyLiquor();
}
```

Anonymous classes

```
public abstract class Store {
      public abstract void welcome(String name);
}
public class LiquorStore implements Store {
     @Override
     public void welcome(String name) { ... }
}
// MAIN
Store store = new Store() {
     public void welcome(String name) {
          System.out.println("Welcome to our liquor store, "
          + name + ". If you are younger than 18, go back
          home!");
          }
};
```

Lambda functions

```
public interface Store {
     public void welcome(String name);
}

// MAIN
Store store = (n) -> {
        System.out.println("Welcome to our liquor store, "
        + n + ". If you are younger than 18, go back home!");
        };
```

Generics (templates)

```
public class GenericExample<T> {
   private T generic;
   public GenericExample(T generic) {
      this.generic = generic;
   }
   public void showType() {
      System.out.println(generic.getClass().getName().toString());
      // We can't use for example .substring() because <T> can be
      anything.
   }
   public T getGeneric() {
      return generic;
   }
}
GenericExample<String> genEx = new GenericExample<>("Hello world!");
genEx.showType(); → java.lang.String  // Out of the class, in this
context we can use a String method with the generic object because the
compiler knows that the generic is a string
System.out.println( genEx.getGeneric().length());
```