Group 4:

- Dilan Shaminda
- Sadaf Alvani
- Server Khalilov
- Tareq Nasif

1. Representation of types

In order to represent types, we are using in our solution the wrapper class **Type**, which in its turn uses implemented by us **MatcherTypes** class in order to distinguish between specific **MJTypes** like **MJTypeInt**, **MJTypeBool**, etc. The **Type** class consists of two fields, which are **TypeChecker** and **String** objects. Both of them are needed in the implementation of **isSubType()** method inside **TypeChecker** class.

2. How do we perform type checking

The type checking in our implementation covers two situations. The first one is responsible for the comparisons of usual objects like **Boolean**, **Integer**, **objects of the same class**, etc. The second one is needed in order to check if one object is the subtype of the another one (which means, it extends the another one). So, in order to cover type checking, when we receive the information of type **MJType**, we are using our class **Type**, which I have described above. Afterwards, we are passing new formed objects of type **Type** to the method **isSubType()** (we have overloaded the implementation of this method, which allows to go inside the really needed implementation in that specific moment).

3. Handling of the different variable scopes

We cover this situation using **Map<Integer, List<String>>** as **history** variable and **List<String>** as **currentScope** variable. Moreover, we always have the counter **count** in order to perform operations on the **history** variable. The usage of the variable **currentScope** (which we use when we enter new scope) is useful, because it allows us to check the previous scope, when we are inside the inner block. Basically, that is our idea of handling variable scopes. In order to check, whether they are used correctly, we perform assertions and check, whether the variable could be used inside the scope.

4. How do we provide analysis information to the translation phase?

- **MethodCall**. For each method call, inside the **Matcher** in the class **TypeCheckVisitor**, we are getting the declaration of the method and store it the variable **method**.
- **NewObject**. As we have the variable **classesInfo**, which stores all the declarations and all the extended classes, inside the **Matcher** of the class **TypeCheckVisitor**, we can get the access to the declaration of the corresponding to the new Object class using method **.getClassName()** of new object and by finding the corresponding class in the **classesInfo**.
- **FieldAccess** and **VarUse**. For each field access, inside the **Matcher** in the class **TypeCheckVisitor**, we are getting the declaration of the variable and store it the variable **varName**. The same applies to the **VarUse**, the declaration of which we store in the **VarDecl** using **case_VarUse** inside the **Matcher**.