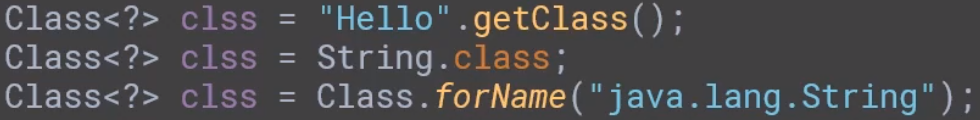
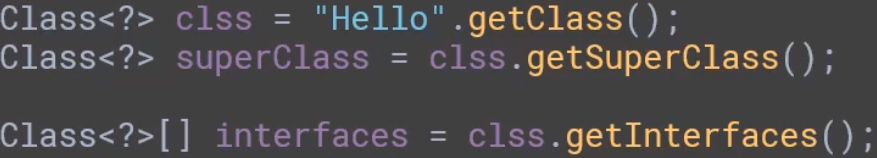
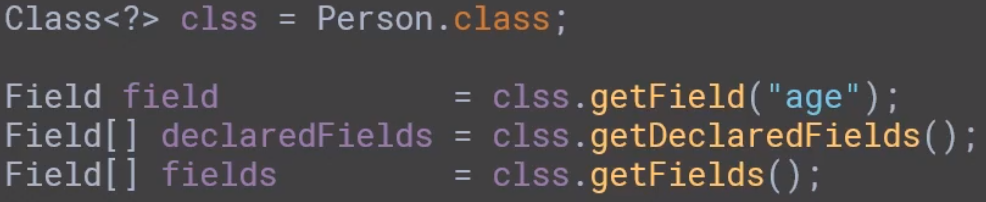
# Three patterns to get a Class instance



# Retrieving informations from Class instance:



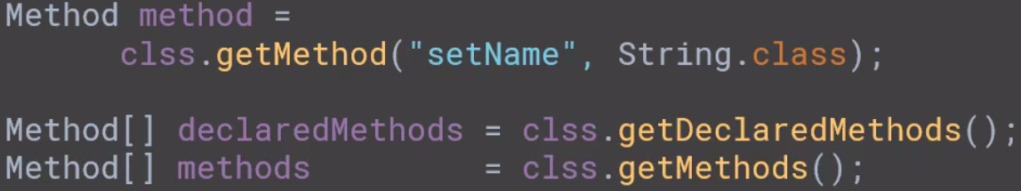
# Get the fields of the class:



getField and getFields retrieve public field(s) but getFields returns public fields of the class and superclasses

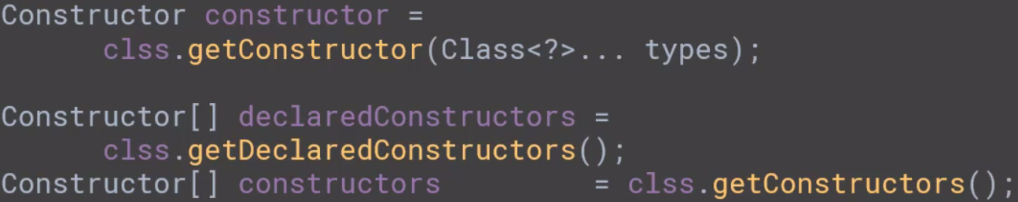
getDeclaredFields returns private fields as well but without the inherited elements

# Get methods of the class



similar meaning to the methods returing fields

# Constructor



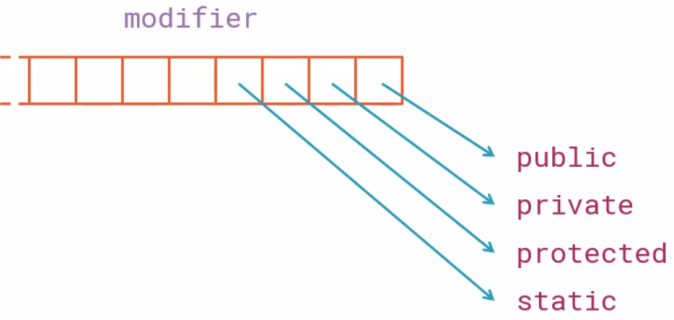
getConstructors() – public constructors in class

getDeclaredConstructors() – all constructors declared in the class

# Modifiers

getModifiers() -> call on Filed, Method and Constructor objects

returns int with 1s on positions according to concrete modifier, example:



can mask with binary number ex:

if ((modifiers & 0b00000001) == 0b00000001) {  
 System.*out*.println("public");  
}  
if ((modifiers & 0b00000010) == 0b00000010) {  
 System.*out*.println("private");  
}  
if ((modifiers & 0b00000100) == 0b00000100) {  
 System.*out*.println("protected");  
}

Better and less prone is to use Modifier factory class and its methods is…. ex:

if (Modifier.*isPublic*(modifiers)) {  
 System.*out*.println("PUBLIC");  
}

# Creating and object metamodel using annotations and reflection

When maping object to database, file etc we need to tell what to save from objects and how to save it. What we need is metadata added to fields of class for example

public class Person {  
  
 @PrimaryKey  
 private int id;  
  
 @Column  
 private String name;

and so on. Annotationas are defined with Runtime retention:

port java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
public @interface PrimaryKey {  
}

Now we can create our metamodel for finding apprpriate fields:

public class Metamodel<T> {  
  
 private final Class<T> clazz;  
  
 public static <T> Metamodel of(Class<T> clazz) {  
 return new Metamodel(clazz);  
 }  
  
 public Metamodel(Class<T> clazz) {  
 this.clazz = clazz;  
 }  
  
 public PrimaryKeyField getPrimaryKey() throws NoSuchFieldException {  
 final Field[] declaredFields = clazz.getDeclaredFields();  
 final Field field = Arrays.*asList*(declaredFields)  
 .stream()  
 .filter(f -> f.getAnnotation(PrimaryKey.class) != null)  
 .findFirst().orElseThrow(() -> new NoSuchFieldException("No primary key field found"));  
  
 return new PrimaryKeyField(field);  
 }

…

And execute it in main method for finding primary key and …

Metamodel<Person> metamodel = Metamodel.*of*(Person.class);  
PrimaryKeyField primaryKeyField = metamodel.getPrimaryKey();  
System.*out*.println("Primary key : " + primaryKeyField.getName() + ", " + primaryKeyField.getType().getSimpleName());

column fields (annotated with @Column)

List<ColumnField> columnFiledList = metamodel.getColumns();