УНИВЕРСИТЕТ ИТМО
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

# Отчёт по лабораторной работе №8
# Вариант 606

Выполнили
Ощепков Артём,
Камакин Кирилл
группа р3102
Преподаватель Письмак А. Е.

Санкт-Петербург 2017

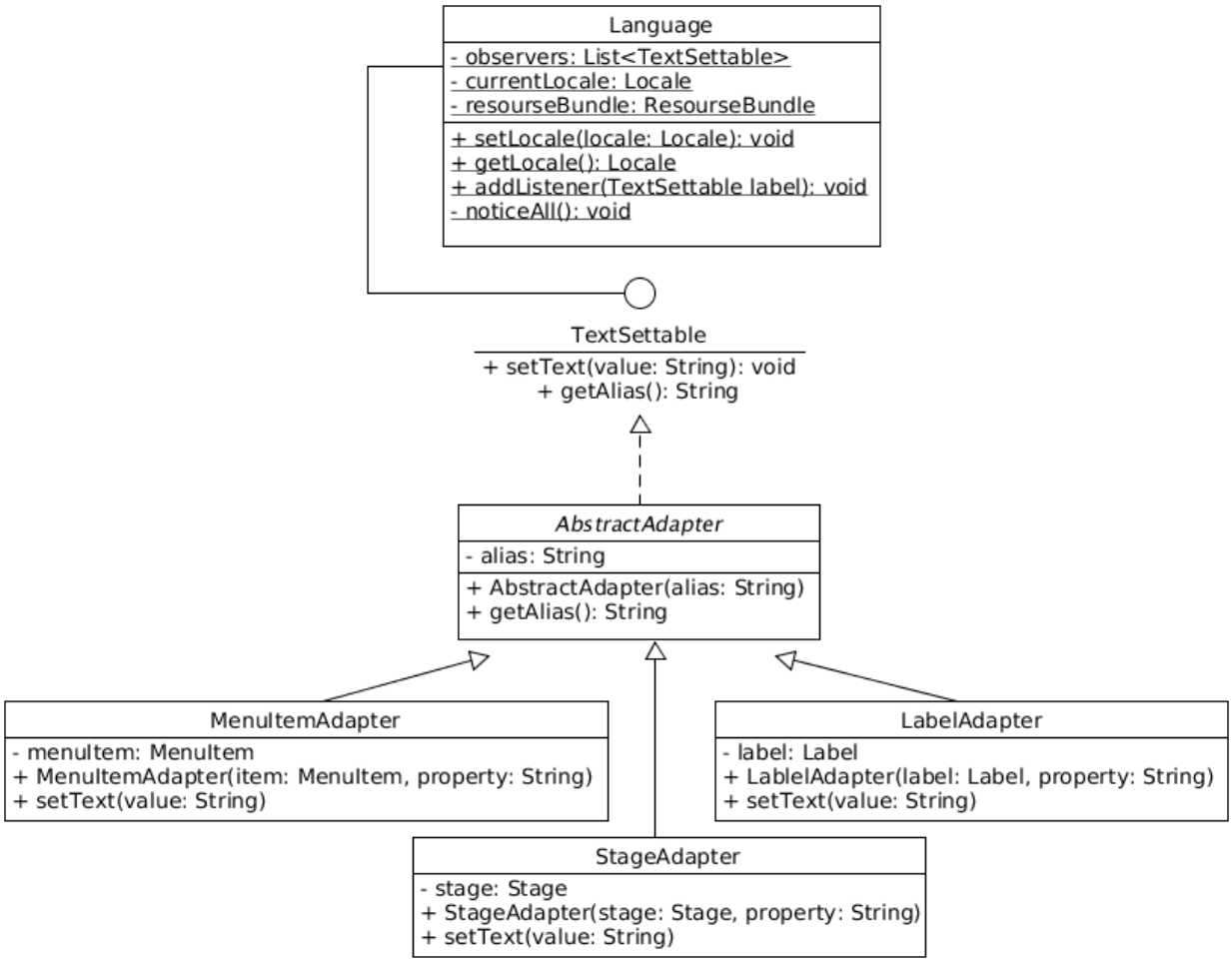
## Текст задания

Доработать программу из лабораторной работы №7 следующим образом:

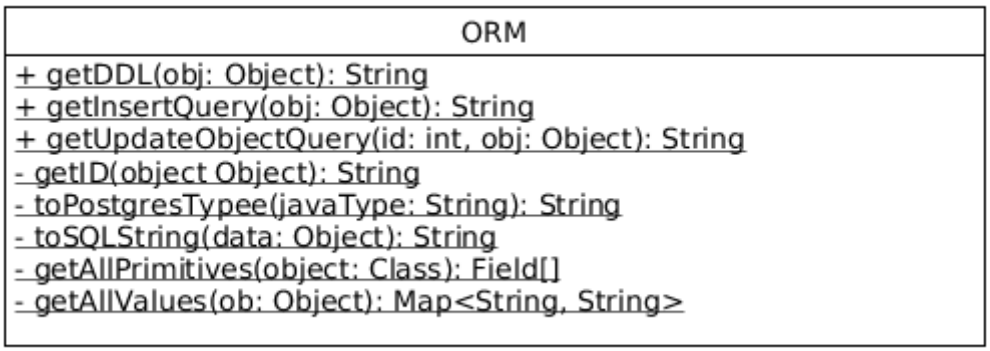
1. В класс, объекты которого хранятся в коллекции, добавить поле типа `java.time.ZonedDateTime`, в котором должны храниться дата и время создания объекта.
2. Графический интерфейс клиентской части должен поддерживать **русский**, **македонский**, **польский** и **английский (Канада)** языки / локали. Должно обеспечиваться корректное отображение чисел, даты и времени в соответствии с локалью. Переключение языков должно происходить без перезапуска приложения. Локализованные ресурсы должны храниться в классе. Сортировку и фильтрацию значений столбцов реализовать с помощью Streams API.
3. Сервер должен обеспечивать простейшую реализацию объектно-реляционного отображения с использованием рефлексии, в том числе создание таблицы базы данных в соответствии с полями объекта и CRUD-запросы.

# Диаграмма классов разработанной программы

## Модуль локализации




## Модуль ORM



# Исходный код:

### Модуль локализации

```
/**
* Класс для интернационализации приложения
* Использование - добавляешь с помощью метода addListener то, у чего
*можно менять текст, устанавливаешь у этого чего-то в качестве ID
*properties name, а *дальше класс всё сделает сам
*/
public class Language {
   static {
      observers = new LinkedList<>();
      resourceBundle = ResourceBundle.getBundle("vt/smt/GUI/languages",new
Locale("ru"));
      currentLocale = Locale.getDefault();
   }
   // Текста, жаждущие меняться
   private static List<TextSettable> observers;
   private static ResourceBundle resourceBundle;
   private static Locale currentLocale;
   public static void setLanguage(Locale locale){
      resourceBundle =
ResourceBundle.getBundle("vt/smt/GUI/languages",locale);
      currentLocale = locale;
         vt.smt.Client.Sender.getInstance().sendCommand(new
                                                      ChangeLocale(currentLocale));
      noticeAll();
   }
   public static String getString(String key){
      return resourceBundle.getString(key);
   }
   public static Locale getLocale(){
      return currentLocale;
   }
   // Юзай этот метод для добавления слушающего текста
   public static void addListener(TextSettable label){
      observers.add(label);
      label.setText(resourceBundle.getString(label.getAlias()));
   }
   private static void noticeAll(){
      observers.forEach(current->{
         if(current == null) // Проблема убывших наблюдателей решается
здесь
            observers.remove(current);
            else
         current.setText(resourceBundle.getString(current.getAlias()));
      });
   }
}
```

```
interface TextSettable{
   void setText(String value);
   String getAlias();
}
/**
 * JavaFx не имеет интерфейса textSettable,
 * поэтому пришлось реализовать много адаптеров
 */
abstract class AbstractAdapter implements TextSettable{
   private String alias;
   public AbstractAdapter(String alias){
      this.alias = alias;
   }
   public String getAlias(){return alias;}
}
class MenuItemAdapter extends AbstractAdapter{
   private MenuItem menuItem;
   public MenuItemAdapter(MenuItem item,String property){
      super(property);
      this.menuItem = item;
      }
   public void setText(String value){
      menuItem.setText(value);
   }
}
class StageAdapter extends AbstractAdapter{
   private Stage stage;
   public StageAdapter(Stage stage,String property){
      super(property);
      this.stage= stage;
   }
   public void setText(String value){
      this.stage.setTitle(value);
   }
}
```

## Модуль ORM

```
public class ORM {
   /**
    * Метод, генерирующий код создания таблицы по заданному обекту
    * @param obj - объект, требующий отображения в бд
    * @return SQL-запрос, создающий требуемую таблицу
    */
   public static String getDDL(Object obj){
      //Строка, в которой постепенно будет формироваться запрос
      StringBuilder query = new StringBuilder("create table ");
      query.append(obj.getClass().getSimpleName() + "(\n ");
      query.append(getID(obj) + " serial primary key,\n ");
         for (Field field : getAllPrimitives(obj.getClass())) {
            query.append(field.getName() + " ");
```

```
            query.append(toPostresType(field.getType().getSimpleName()) +",\n " );
          }
        // Удаление последней лишней запятой
        query.deleteCharAt(query.length()-3);
        query.append(");");
       return query.toString();
    }
    /**
     * Метод формирования запроса к БД на вставку данного объекта
     * Имя класса считается названием таблицы
     */
    public static String getInsertQuery(Object obj){
      StringBuilder query = new StringBuilder(
            "insert into " + obj.getClass().getSimpleName() + " " );
      Map<String,String> values = getAllValues(obj);
      query.append(values.keySet().toString() + " values " + values.values() +
';');
      return query.toString().replace('[','(').replace(']',')');
    }
    /**
    *  Метод, считывающий все геттеры, которые класс предоставляет и
составляющий из них
     * Ассоциативную карту (название геттера (или метода is) без
префикса get, - значение поля)
     * Не боится иерархии и содержания непримитивных типов
     * @param obj Наш подопечный - передаём не Класс, а экземпляр,
потому что нуждаемся в данных
     *
    **/
    private static Map<String,String> getAllValues(Object obj) {
      Map<String,String> result = new HashMap<>(); // Здесь конструируется
результат работы метода
      for (Method method : obj.getClass().getMethods()) {
        try { // Просто берём все доступные геттеры и вызываем их
          if(method.getName().startsWith("get") && !
method.getName().equals("getClass"))

result.put(method.getName().substring(3),toPSQLString(method.invoke(obj)));
          if(method.getName().startsWith("is"))
            result.put(method.getName(),toPSQLString(method.invoke(obj)));
        }catch (Exception e){
          System.out.println(e.getMessage());
          e.printStackTrace();
        }
      }
      // Поля, являющиеся непримитивными типами, тоже следует
обработать. Применяется рекурсия
      for (Field field :
obj.getClass().asSubclass(obj.getClass().getSuperclass()).getDeclaredFields()) {
        field.setAccessible(true);
        try { // Примитивный тип или нет определяеся вот таким вот
```

```
образом
            if (toPostresType(field.getType().getSimpleName()) == null)
               result.putAll(getAllValues(field.get(obj)));
          }catch (Exception e){
            e.printStackTrace();
          }
       }
       return result;
    }

    /**
     * Преобразование типа для видения sql
     * (например, значения булевских типов должны быть заключёны в
кавычки)
     */
    private static String toPSQLString(Object data){
       if(data.getClass().getSimpleName().equals("Boolean") ||
         data.getClass().getSimpleName().equals("boolean") ||
         data.getClass().getSimpleName().equals("String") )
        return '\'' + data.toString() + '\'';
       if(data instanceof ZonedDateTime){
         return '\'' +
((ZonedDateTime)data).format(DateTimeFormatter.ofPattern("yyy-MM-dd")) +
'\'';
       }
     return data.toString();
    }
    /**
     * Метод, умеющий считывать все примитивные поля объекта,
     * учитывая иерхархию и композицию непримитивных типов
     * @param obj - Класс исследуемого объекта
    **/
    private static Field[] getAllPrimitives(Class obj){
       LinkedList<Field> fields = new LinkedList<>(); // Для формирования
результатов
       for (Field field : obj.getDeclaredFields()) {
         //Если мы знаем, как преобразовать этот тип к типу данных постгрес
         if (toPostresType(field.getType().getSimpleName()) != null)
            fields.add(field); // Просто добавляем поле
         else // Иначе это не примитив, и его нужно обработать таким же
способом
            fields.addAll(Arrays.asList(getAllPrimitives(field.getType())));
       }
       // Если у объекта есть родительский класс ( и не Object)
       if(obj.getSuperclass() != Object.class) // Делаем с ним то же самое
         fields.addAll(Arrays.asList(getAllPrimitives(obj.getSuperclass())));
       return fields.toArray(new Field[0]);
    }
    // Таблица преобразования типов данных Java в типы данных Postgres
    private static String toPostresType(String javaType){
       switch (javaType) {
```

```
        case "String":
          return "varchar(80)";
        case "int":
          return "integer";
        case "long":
          return "integer";
        case "double":
          return "real";
        case "boolean":
          return "boolean";
        case "ZonedDateTime":
          return "date";
      }
      return null;
    }
/**
* Как назвать поле первичного ключа решается здесь
**/
   private static String getID(Object object){
     return object.getClass().getSimpleName() + "_id";
   }
}
```

# Выводы:

*Тёма:*

Дописан крайний отчёт по проге. Около пятидесяти классов содержится в итоговом приложениии , символов переноса строки во всех выживших исходников ровно 2389, порядка 80-ти коммитов залетело на гитхаб..

Но это всё цифры.

Курс по программированию был самым увлекательным предемтом прошедшего года.

Спасибо за это. Свою крайнюю лабораторную работу я собирался делать при свечах, но соседи напомнили, что в МСГ за такие радости могут и выселить

Чувствую, будто нахожусь в дестком лагере на огоньке в ночь перед отъездом

Скорее, перед броском в пропасть, потому что скоро сессия

До новых встреч!

*Кирилл:*

В свою очередь, я ближе познакомился с такой сложной технологий как ORM, попробовал создать своё ORM. Кроме того, мне представилась возможность поработать в команде, написать совместный код. Запоминающимся был первый локализаторский опыт. Тамада хороший и конкурсы весёлые