Student: Tri Hong Nguyen - 553719

0.1 Exercise 1.

Code 1: Annotation Structure.

```
1  public class AnnotationStructure {
2  private String AnnotationName;
3  private Map<String, String> KeyValues = new HashMap<String, String>();
4  public AnnotationStructure(String InputName,Map<String, String> InputKeyValues){
5   this.AnnotationName = InputName;  this.KeyValues = InputKeyValues;
6  }
7  public AnnotationStructure(AnnotationStructure A){
8   this.AnnotationName = A.AnnotationName;  this.KeyValues = A.KeyValues;
9  }
10 public String getValue(String K){return this.KeyValues.get(K);}
11 }
```

Code 2: Field Structure.

```
1  public class FdStr {
2  protected AnnotationStructure Annotation;
3  protected ArrayList<Type> Type = new ArrayList<>();
4  private String Name;
5  public FdStr(AnnotationStructure A, ArrayList<Type> T, String V){
6   this.Annotation = A;  this.Type = T;  this.Name = V;
7  }
8  public FdStr(FdStr A){
9   this.Annotation = A.Annotation;  this.Type = A.Type;  this.Name = A.Name;
10 }
11 public AnnotationStructure printAnnotaion(){return this.Annotation;}
12 public ArrayList<Type> printType(){return this.Type;}
13 public String printName(){return this.Name;}
14 public String printJAVA(){
15  if(this.Type.size()==1){
16   switch (Type.get(0).printType()) {
17   case "String":
18    String len = this.Annotation.getValue("length");
19    if (len.contains(".")) return "VARCHAR("+(len.split("\\.")[0]+")";
20    return "VARCHAR(" + len + ")";
21   case "Integer":
22    if (this.Annotation.getValue("name").equals("id")) return "INT NOT NULL
       PRIMARY KEY";
23    else return "INT";
24   default:
25    if(Scanner.sb.get(Type.get(0).printType()).equals(Scanner.TKT.NEWTYPE) &&
       this.Annotation.getValue("target").equals(Type.get(0).printType()))
26     return "FOREIGN KEY REFERENCES " + Type.get(0).printType();
27    return null;
28   }
29  } else return null;
30 }}
```

Code 3: Interface Structure.

```
1  public class InterStr {
2  private AnnotationStructure Annotation;
3  private String Name;
4  private ArrayList<FdStr> Inside;
5  public InterStr(AnnotationStructure A, String N, ArrayList<FdStr> I){
6   this.Annotation = A; this.Name = N; this.Inside = I;
7  }
8  public String printName(){return this.Name;}
9  public AnnotationStructure printAnnotation(){return this.Annotation;}
10 public ArrayList<FdStr> printField(){return this.Inside;}
```

```
}
```

0.2 Exercise 2.

Code 4: Scanner Class.

```
public class Scanner{
public enum TKT {INTERFACE, LIST, PUBLIC, AT, KEY, VALUE, ASSIGN, COMMA, NEWTYPE, NULL,
    DQUOTE, SEMICOLON, OPEN_CURLYBRACKET, CLOSE_CURLYBRACKET, OPEN_BRACKET,
    CLOSE_BRACKET, INT, STRING, EOF, EOA, NO_T, BG, LS, NAME, NUMBER, };
protected static Hashtable<String, TKT> sb=new Hashtable<String, TKT>();
private String BackWord;
private String CurrentWord;
private String FrontWord;
private StreamTokenizer addNew;
public Scanner(Reader read){
 addNew = new StreamTokenizer(read);
 addNew.wordChars('a','z');        addNew.wordChars('A', 'Z');
 addNew.eolIsSignificant(false); addNew.parseNumbers();
 char[] SpecialChar={'{','}','(',')',';','>','<','=','"','@',','};
 for (char charac:SpecialChar) addNew.ordinaryChar(charac);
 sb.put("Integer",TKT.INT);    sb.put("String",TKT.STRING);
 sb.put("List",TKT.LIST);      sb.put("interface",TKT.INTERFACE);
 sb.put("public",TKT.PUBLIC); sb.put("@", TKT.AT);
}
public TKT nextToken(){
 try {
  BackWord = addNew.sval;  int next = addNew.nextToken();
  if(addNew.ttype == StreamTokenizer.TT_WORD) CurrentWord = addNew.sval;
  else if(addNew.ttype == StreamTokenizer.TT_NUMBER)  CurrentWord = String.
      valueOf(addNew.nval);
  switch(next){
  case StreamTokenizer.TT_EOF: return TKT.EOF;
  case StreamTokenizer.TT_WORD:int t=addNew.nextToken(); FrontWord=addNew.sval;
   switch (t) {
    case '"': addNew.pushBack();  return TKT.VALUE;
    case '=': addNew.pushBack();  return TKT.KEY;
    case StreamTokenizer.TT_WORD:
     if(CurrentWord.equals("interface")){
      addNew.pushBack();
      sb.put(FrontWord, TKT.NEWTYPE);
      return (sb.get(CurrentWord));
     }else{
      if(sb.get(CurrentWord)==null && sb.get(FrontWord)==null){
       addNew.pushBack();
       sb.put(CurrentWord, TKT.NEWTYPE);
       return (sb.get(CurrentWord));
      }
     }
    default:
     addNew.pushBack();
     if(sb.get(CurrentWord)==null) sb.put(CurrentWord , TKT.NAME);
     return (sb.get(CurrentWord));
    }
  case StreamTokenizer.TT_NUMBER: return TKT.VALUE;
  case '{': return TKT.OPEN_CURLYBRACKET;
  case '}': return TKT.CLOSE_CURLYBRACKET;
  case ';': return TKT.SEMICOLON;
  case ',': return TKT.COMMA;
  case '(': return TKT.OPEN_BRACKET;
  case ')': return TKT.CLOSE_BRACKET;
  case '=': return TKT.ASSIGN;
  case '>': return TKT.BG;
  case '<': return TKT.LS;
  case '"': return TKT.DQUOTE;
```

```
57    case '@': return TKT.AT;
58    default : return TKT.NO_T;
59    }
60   } catch (IOException e){e.printStackTrace();return TKT.EOF;}
61  }
62  public String getTokenval(){return this.CurrentWord;}
63  }
```

Code 5: Parser Class.

```
1   public class Parser {
2   private Scanner scan;
3   private Scanner.TKT lookahead;
4   ArrayList<InterStr> IL = new ArrayList<>();
5   private void match(Scanner.TKT t) throws SyntaxException{
6    if(lookahead!=t) throw new SyntaxException("Expected␣"+t);lookahead=scan.
        nextToken();
7   }
8   private void expect(Scanner.TKT t){
9    if(lookahead!=t) throw new SyntaxException("Failed:expected␣"+t);
10  }
11  public ArrayList<InterStr> parseMain(Reader r){
12   scan =new Scanner(r);
13   lookahead = scan.nextToken();
14   return parseInterfaceList(IL);
15  }
16  public ArrayList<InterStr> parseInterfaceList(ArrayList<InterStr> IL){
17   switch (lookahead) {
18   case EOF: return null;
19   case AT:
20    IL.add(parseInterface());
21    if(lookahead == Scanner.TKT.EOF) return IL;
22    else return parseInterfaceList(IL);
23   default: return null;
24   }
25  }
26  public InterStr parseInterface(){
27   switch (lookahead) {
28   case AT:
29    AnnotationStructure A = new AnnotationStructure(parseAnnotaion());
30    match(Scanner.TKT.PUBLIC);
31    match(Scanner.TKT.INTERFACE);
32    expect(Scanner.TKT.NEWTYPE);
33    String N = (String) scan.getTokenval();
34    match(Scanner.TKT.NEWTYPE);
35    match(Scanner.TKT.OPEN_CURLYBRACKET);
36    ArrayList<FdStr> F = new ArrayList<>();
37    F = FieldList(F);
38    F.get(0).printName();
39    match(Scanner.TKT.CLOSE_CURLYBRACKET);
40    return new InterStr(A,N,F);
41   default: return null;
42   }
43  }
44  private AnnotationStructure parseAnnotaion(){
45   switch (lookahead) {
46   case CLOSE_BRACKET: return null;
47   default:
48    match(Scanner.TKT.AT);
49    expect(Scanner.TKT.NAME);
50    String name = (String) scan.getTokenval();
51    match(Scanner.TKT.NAME);
52    match(Scanner.TKT.OPEN_BRACKET);
53    Map<String,String> KeyValues = new HashMap<String, String>();
```

```
54    KeyValues = KeyValuesList(KeyValues);
55    match(Scanner.TKT.CLOSE_BRACKET);
56    return new AnnotationStructure(name,KeyValues);
57   }
58  }
59  private Map<String,String> KeyValuesList(Map<String,String> KeyValues){
60   switch (lookahead) {
61   case CLOSE_BRACKET:     return null;
62   default:
63    expect(Scanner.TKT.KEY);
64    String K = (String) scan.getTokenval();
65    match(Scanner.TKT.KEY);
66    match(Scanner.TKT.ASSIGN);
67    match(Scanner.TKT.DQUOTE);
68    String V;
69    switch (lookahead) {
70    case NUMBER:
71     expect(Scanner.TKT.NUMBER);
72     V = (String) scan.getTokenval();
73     match(Scanner.TKT.NUMBER);
74     break;
75    case VALUE:
76     expect(Scanner.TKT.VALUE);
77     V = (String) scan.getTokenval();
78     match(Scanner.TKT.VALUE);
79     break;
80    default: V= null; break;
81    }
82    match(Scanner.TKT.DQUOTE);
83    KeyValues.put(K, V);
84    if (lookahead == Scanner.TKT.COMMA){match(Scanner.TKT.COMMA);
85     return KeyValuesList(KeyValues);}
86    else return KeyValues;
87   }
88  }
89  private ArrayList<FdStr> FieldList(ArrayList<FdStr> F){
90   switch(lookahead){
91   case CLOSE_CURLYBRACKET: return F;
92   default:
93    FdStr temp = new FdStr(parseField());
94    F.add(temp);
95    return (lookahead == Scanner.TKT.AT) ? FieldList(F) : F;
96   }
97  }
98  private FdStr parseField(){
99   switch(lookahead){
100  case CLOSE_CURLYBRACKET: return null;
101  default:
102   AnnotationStructure A = new AnnotationStructure(parseAnnotaion());
103   ArrayList<Type> T = new ArrayList<>();
104   T = TypeList(T);
105   expect(Scanner.TKT.NAME);
106   String N=(String)scan.getTokenval();
107   match(Scanner.TKT.NAME);
108   match(Scanner.TKT.SEMICOLON);
109   return new FdStr(A,T,N);
110  }
111 }
112 private ArrayList<Type> TypeList(ArrayList<Type> T){
113  Type t = new Type(parseType());
114  if(!t.printType().equals("NULL"))      T.add(t);
115  switch (lookahead) {
116  case CLOSE_CURLYBRACKET: return null;
```

```
 case NAME: return T;
 case LS:
  match(Scanner.TKT.LS);
  T = TypeList(T);
  return T;
 case BG:
  match(Scanner.TKT.BG);
  return (lookahead == Scanner.TKT.NAME)? T : TypeList(T);
 default:  return null;
 }
}
private Type parseType(){
 Scanner.TKT T=lookahead;
 switch (lookahead) {
 case NEWTYPE:
  for(int i=0;i<Scanner.sb.size();i++){
   if(Scanner.sb.get(scan.getTokenval()).equals(Scanner.TKT.NEWTYPE)){
    match(Scanner.TKT.NEWTYPE);
    return new Type(scan.getTokenval().toString());
   }
  }
  match(Scanner.TKT.NEWTYPE);
  return new Type(T);
 case INT: case STRING: case LIST: match(lookahead); return new Type(T);
 default: return new Type(Scanner.TKT.NULL);
 }
}}
```

Code 6: Type Class.

```
public class Type{
String type;
public Type(Scanner.TKT in){this.type = in.toString();}
public Type(Type a){this.type = a.type;}
public Type(String a){this.type = a;}
public String printType(){
 for(Map.Entry<String, Scanner.TKT> entry : Scanner.sb.entrySet()){
  if(this.type.equals(entry.getValue().toString())){
   this.type = entry.getKey(); break;
  }
}return this.type;
}
public Boolean checkType(){
 Scanner.TKT check = Scanner.sb.get(this.type);
 if(this.type.equals(Scanner.TKT.NEWTYPE.toString())){
  for(int i=0; i<Scanner.sb.size();i++)
   if(check.equals(Scanner.TKT.NEWTYPE)) return true;
   else return false;
 }else return true;
 return null;
}}
```

0.3 Exercise 3.

Code 7: Java Generator.

```
public class JavaGenerator {
protected ArrayList<InterStr> data;
public JavaGenerator(ArrayList<InterStr> in) {this.data = in;}
public void printNewType() throws Exception{
 for(InterStr Inter : this.data){
  File file1 = new File("src/" + Inter.printName() +".java");
  ArrayList<FdStr> ListofFields = new ArrayList<>(Inter.printField());
  ArrayList<String> lines = new ArrayList<>();
  lines.add("");
```

```
10    lines.add("public class " + Inter.printName() + "{");
11    ArrayList<String> Constructor = new ArrayList<>();
12    String linesConstructor = "public " + Inter.printName() + "(";
13    Constructor.add(linesConstructor);
14    for(FdStr Field : ListofFields){
15     ArrayList<Type> type = new ArrayList<Type>(Field.printType());
16     String lineField = "protected ";
17     for(int itype=0;itype<type.size();itype++){
18     if(type.get(itype).printType().equals("List"))
19      lines.set(0, "import java.util.List;" + "\nimport java.util.ArrayList;");
20      if(Scanner.sb.get(type.get(itype).printType().toString()).equals(Scanner.TKT
            .NEWTYPE) && type.size()==1)
21       lines.set(1,"public class " + Inter.printName() + " extends " + type.get(
            itype).printType().toString() +"{");
22      lineField += type.get(itype).printType();
23      linesConstructor += type.get(itype).printType();
24      if(itype+1<type.size()){
25       lineField += "<";  linesConstructor+= "<";
26      }
27     }
28     for(int closeBG=0;closeBG<type.size()-1;closeBG++){
29      lineField += ">";  linesConstructor += ">";
30     }
31     lineField += " " + Field.printName() + ";";
32     lines.add(lineField);
33     if (ListofFields.get(ListofFields.size()-1).equals(Field))
34      linesConstructor += " " + Field.printName() + "_temp";
35     else linesConstructor += " " + Field.printName() + "_temp,";
36      Constructor.add("this."+Field.printName()+" ="+Field.printName()+"_temp;");
37    }
38    lines.add("public " + Inter.printName() + "(){}");
39    linesConstructor += "){";
40    Constructor.add("}}"); Constructor.set(0, linesConstructor);
41    try {
42     file1.createNewFile();  FileWriter writer = new FileWriter(file1);
43     for(String f: lines) writer.write(f+"\n");
44     for(String f: Constructor) writer.write(f+"\n");
45     writer.flush();  writer.close();
46    } catch (IOException e) {e.printStackTrace();}
47   }
48  }}
```

Code 8: SQL Generator.

```
1   public class SQLGenerator extends JavaGenerator{
2   public ArrayList<String> lines = new ArrayList<>();
3   public SQLGenerator(ArrayList<InterStr> in) throws IOException {
4    super(in);
5    String NameFile = "SQLGenerator.sql";
6    File file = new File(NameFile); file.createNewFile();
7    FileWriter writer = new FileWriter(file);
8    for(InterStr Inter : data){
9     lines.add("CREATE TABLE " + Inter.printAnnotation().getValue("name") + " ( id
          INT NOT NULL PRIMARY KEY");
10    lines.set(lines.size()-1, lines.get(lines.size()-1).split("\\,")[0]);
11    lines.add(");\n");
12   }
13   try {
14    for(String f: lines) writer.write(f+"\n");
15    writer.flush(); writer.close();
16   } catch (IOException e) {e.printStackTrace();}
17  }
18  public String printSQL(FdStr f){
19   if(f.Type.size()==1){
```

```
20  switch (f.Type.get(0).printType()) {
21  case "String": String len = f.Annotation.getValue("length");
22   if (len.contains(".")) return "VARCHAR("+(len.split("\\."))[0]+")";
23   return "VARCHAR(" + len + ")";
24  case "Integer": if(f.Annotation.getValue("name").equals("id")) return "INT";
25  default:
26   if(Scanner.sb.get(f.Type.get(0).printType()).equals(Scanner.TKT.NEWTYPE) && f.
        Annotation.getValue("target").equals(f.Type.get(0).printType())){
27    String name = f.Annotation.getValue("name");
28    String target = f.Annotation.getValue("target");
29    String out = "INT," + "ADD FOREIGN KEY (" + name + ") REFERENCES `" + target.
         toLowerCase() + "`(`id`)";
30    return out;
31   } return null;
32  }
33 }else return null;
34 }
35 public void printNewType(){
36  String NameFile = "SQLGenerator.sql";
37  for(InterStr Inter : data){
38   ArrayList<FdStr> ListofFields = new ArrayList<>(Inter.printField());
39   lines.add("ALTER TABLE `" + Inter.printAnnotation().getValue("name") + "` ");
40   for(int ifield=1; ifield<ListofFields.size();ifield++){
41    String lineField = null;
42    if (printSQL(ListofFields.get(ifield))!= null){
43     lineField = "ADD COLUMN " + ListofFields.get(ifield).printAnnotaion().
          getValue("name") + " ";
44     lineField += printSQL(ListofFields.get(ifield));
45     if(ifield+1<ListofFields.size()) lineField += ",";
46     lines.add(lineField);
47    }else lines.set(lines.size()-1,lines.get(lines.size()-1).split("\\,")[0]);
48   }
49   lines.add(";\n");
50  }
51  try {
52   File file = new File(NameFile); file.createNewFile();
53   FileWriter writer = new FileWriter(file);
54   for(String f: lines) writer.write(f+"\n");
55   writer.flush();  writer.close();
56  } catch (IOException e) {e.printStackTrace();}
57 }}
```

0.4 Exercise 4.

Code 9: IEntityManger Class.

```
1  public class IEntityManagerClass<T> implements IEntityManager<T>{
2  protected Class<T> type;
3  protected int waitPublisher = 0;
4  public IEntityManagerClass(Class<?> tem){this.type = (Class<T>) tem;}
5  public IEntityManagerClass(){}
6  @Override
7  public void persist(T entity)  {
8   File file = new File("SQLGenerator.sql");
9   StringBuilder sb = new StringBuilder();
10  Class<?> thisClass = null;
11  try {
12   file.createNewFile();  FileWriter writer = new FileWriter(file,true);
13   thisClass = Class.forName(entity.getClass().getName());
14   java.lang.reflect.Field[] aClassFields = thisClass.getDeclaredFields();
15   sb.append("INSERT INTO " + entity.getClass().getSimpleName() + " VALUES(");
16   for(java.lang.reflect.Field f : aClassFields){
17    if(f.get(entity)!= null){
18     if(f.getType().getSimpleName().equals("String")){
19      if(f==aClassFields[aClassFields.length-1]) sb.append(f.get(entity));
```

```
20        else  sb.append("\'" + f.get(entity) + "\'"   + ", ");
21       }else if(f.getType().getSimpleName().equals("Integer")){
22        if(f==aClassFields[aClassFields.length-1]) sb.append(f.get(entity));
23        else sb.append(f.get(entity) + ", ");
24       }else{
25         java.lang.reflect.Field[] aClassFields2 = f.getType().getDeclaredFields();
26         for(java.lang.reflect.Field f2 : aClassFields2)
27          if(f2.getName().equals("id")) sb.append(f2.get(f.get(entity)));
28       }
29      }else sb.append("NULL");
30     }
31     sb.append(");");
32     writer.write("\n" + sb.toString());  writer.flush();  writer.close();
33   } catch (Exception e) {e.printStackTrace();}
34  }
35  @Override
36  public void remove(T entity) {
37   StringBuilder sb = new StringBuilder();
38   Class<?> thisClass = null;
39   File file = new File("SQLGenerator.sql");
40   try {
41    file.createNewFile(); FileWriter writer = new FileWriter(file,true);
42    thisClass = Class.forName(entity.getClass().getName());
43    java.lang.reflect.Field[] aClassFields = thisClass.getDeclaredFields();
44    sb.append("DELETE FROM " + entity.getClass().getSimpleName().toLowerCase() + "
        WHERE ");
45    for(java.lang.reflect.Field f : aClassFields)
46     if(f.get(entity)!= null && f.getName().equals("id"))
47      sb.append(f.getName() + " = " + f.get(entity));
48     else continue;
49    sb.append(";");
50    writer.write("\n" + sb.toString());  writer.flush();  writer.close();
51  } catch (Exception e) {e.printStackTrace();}
52  }
53  @Override
54  public T find(Object pk) {
55   T out = null;
56   try {Connection con = DriverManager.getConnection(url,username,password);
57    out = type.newInstance();
58    String q = ("SELECT * FROM " + type.getName().toLowerCase() + " WHERE id = " +
         pk + ";");
59    java.sql.PreparedStatement st = con.prepareStatement(q);
60    ResultSet result = st.executeQuery();
61    ResultSetMetaData metaData = result.getMetaData();
62    if(result.next()){
63     java.lang.reflect.Field[] ListFields = type.getDeclaredFields();
64     int specialPoision = -1;  int icol = 1;
65     while(icol<=metaData.getColumnCount()){
66      if((ListFields[icol-1].getType().getSimpleName().equals("Integer") &&
          metaData.getColumnTypeName(icol).equals("INT")) || (ListFields[icol-1].
          getType().getSimpleName().equals("String") && metaData.getColumnTypeName(
          icol).equals("VARCHAR")))
67       ListFields[icol-1].set(out,result.getObject(icol));
68      else if(metaData.getColumnTypeName(icol).toString().toUpperCase().equals("
          INT") && waitPublisher == 0){
69       specialPoision = icol;
70       Class<?> tem = ListFields[icol-1].getType();
71       IEntityManagerClass<T> a = new IEntityManagerClass<T>(tem);
72       ListFields[icol-1].set(out,a.find(result.getObject(icol)));
73      }else if(metaData.getColumnTypeName(icol).toString().toUpperCase().equals("
          INT") && waitPublisher == 1)
74        ListFields[icol-1].set(out,null);
75       else specialPoision = icol;
```

```
76      icol++;
77      }
78      if(specialPoision == -1)      specialPoision = icol-1;
79      if(metaData.getColumnCount()<ListFields.length){
80       String[] Types_ = ListFields[specialPoision].getGenericType().toString().
             split("\\W");
81       String TempClass = Types_[Types_.length-1];
82       Class<?> tem2 = Class.forName(TempClass);
83       java.lang.reflect.Field[] TempClassFields = tem2.getDeclaredFields();
84       String que ="select *from "+tem2.getSimpleName().toLowerCase()+" where "+
             tem2.getSimpleName().toLowerCase()+"."+TempClassFields[specialPoision].
             getName()+" = "+pk+";";
85       java.sql.PreparedStatement secConnec = con.prepareStatement(que);
86       ResultSet secResult = secConnec.executeQuery();
87       List<T> ListBook = new ArrayList<>();
88       while(secResult.next()){
89        IEntityManagerClass<T> retrBook = new IEntityManagerClass<T>(tem2);
90        retrBook.waitPublisher = 1;
91        ListBook.add(retrBook.find(secResult.getObject(1)));
92       }
93       ListFields[icol-1].set(out,ListBook);
94       for(int iT=0; iT<ListBook.size();iT++){
95        java.lang.reflect.Field[] ListField_elemBook = tem2.getDeclaredFields();
96        for(java.lang.reflect.Field f : ListField_elemBook)
97         if(f.getType().getSimpleName().toString().equals(type.getName())) f.set(
              ListBook.get(iT),out);
98       }
99       ListFields[icol-1].set(out,ListBook);
100     }
101    }else return null;
102   } catch (Exception e) {e.printStackTrace();}
103  return (T) out;
104  }
105  @Override
106  public Query<T> createQuery(String query) {
107   Query<T> out = new Query<>(query, type);
108   return out;
109  }}
```

0.5 Exercise 5.

Code 10: Query Class.

```
1   public class Query<T> implements IQuery<T> {
2   private Class<T> typeOfClass;
3   private String query;
4   protected Query(String q,Class<T> A) {
5    this.query = q; this.typeOfClass = A;
6   }
7   @Override
8   public List<T> getResultList() {
9    List<T> out = new ArrayList<T>();
10   IEntityManagerClass<T> retr = new IEntityManagerClass<T>(typeOfClass);
11   try {Connection con = DriverManager.getConnection(url,username,password);
12    String q = ("SELECT *FROM " + typeOfClass.getName().toLowerCase() + ";");
13    java.sql.PreparedStatement st = con.prepareStatement(q);
14    ResultSet result = st.executeQuery();
15    while(result.next()) out.add(retr.find(result.getObject(1)));
16   } catch (SQLException e) {e.printStackTrace();}
17   return out;
18  }
19  @Override
20  public void execute() {
21   try{Connection con = DriverManager.getConnection(url,username,password);
22    java.sql.PreparedStatement st = con.prepareStatement(query);
```

```
23    st.execute();
24   } catch (Exception e) {e.printStackTrace();}
25  }}
```

0.6 Exercise 6.