Student: Tri Hong Nguyen - 553719

0.1 Exercise 1.

Code 1: Annotation Structure.

```java
public class AnnotationStructure {
private String AnnotationName;
private Map<String, String> KeyValues = new HashMap<String, String>();
public AnnotationStructure(String InputName,Map<String, String> InputKeyValues){
 this.AnnotationName = InputName;  this.KeyValues = InputKeyValues;
}
public AnnotationStructure(AnnotationStructure A){
 this.AnnotationName = A.AnnotationName;  this.KeyValues = A.KeyValues;
}
public String getValue(String K){return this.KeyValues.get(K);}
}
```

Code 2: Field Structure.

```java
public class FdStr {
protected AnnotationStructure Annotation;
protected ArrayList<Type> Type = new ArrayList<>();
private String Name;
public FdStr(AnnotationStructure A,ArrayList<Type> T,String V){
 this.Annotation = A;  this.Type = T;  this.Name = V;
}
public FdStr(FdStr A){
 this.Annotation = A.Annotation;  this.Type = A.Type;  this.Name = A.Name;
}
public AnnotationStructure printAnnotaion(){return this.Annotation;}
public ArrayList<Type> printType(){return this.Type;}
public String printName(){return this.Name;}
public String printJAVA(){
 if(this.Type.size()==1){
  switch (Type.get(0).printType()) {
  case "String":
   String len = this.Annotation.getValue("length");
   if (len.contains(".")) return "VARCHAR("+(len.split("\\."))[0]+")";
   return "VARCHAR(" + len + ")";
  case "Integer":
   if (this.Annotation.getValue("name").equals("id")) return "INT␣NOT␣NULL␣PRIMARY␣KEY";
   else return "INT";
  default:
   if(Scanner.sb.get(Type.get(0).printType()).equals(Scanner.TKT.NEWTYPE) && this.Annotation.getValue("target").equals(Type.get(0).printType()))
    return "FOREIGN␣KEY␣REFERENCES␣" + Type.get(0).printType();
   return null;
  }
 } else return null;
}}
```

Code 3: Interface Structure.

```java
public class InterStr {
private AnnotationStructure Annotation;
private String Name;
private ArrayList<FdStr> Inside;
public InterStr(AnnotationStructure A, String N, ArrayList<FdStr> I){
 this.Annotation = A; this.Name = N; this.Inside = I;
}
public String printName(){return this.Name;}
public AnnotationStructure printAnnotation(){return this.Annotation;}
public ArrayList<FdStr> printField(){return this.Inside;}
}
```

0.2 Exercise 2.

Code 4: Scanner Class.

```
public class Scanner{
private String BackWord;
private String CurrentWord;
private String FrontWord;
private StreamTokenizer addNew;
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>();
public String getTokenval(){return this.CurrentWord;}
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;
   case '@': return TKT.AT;
   default : return TKT.NO_T;
   }
 } catch (IOException e){e.printStackTrace();return TKT.EOF;}
}}
```

Code 5: Parser Class.

```
public class Parser {
private Scanner scan;
private Scanner.TKT lookahead;
ArrayList<InterStr> IL = new ArrayList<>();
private void match(Scanner.TKT t) throws SyntaxException{
 if(lookahead!=t) throw new SyntaxException("Expected␣"+t);lookahead=scan.
     nextToken();
}
private void expect(Scanner.TKT t){
 if(lookahead!=t) throw new SyntaxException("Failed:expected␣"+t);
}
public ArrayList<InterStr> parseMain(Reader r){
 scan =new Scanner(r);
 lookahead = scan.nextToken();
 return parseInterfaceList(IL);
}
public ArrayList<InterStr> parseInterfaceList(ArrayList<InterStr> IL){
 switch (lookahead) {
 case EOF: return null;
 case AT:
  IL.add(parseInterface());
  if(lookahead == Scanner.TKT.EOF) return IL;
  else return parseInterfaceList(IL);
 default: return null;
 }
}
public InterStr parseInterface(){
 switch (lookahead) {
 case AT:
  AnnotationStructure A = new AnnotationStructure(parseAnnotaion());
  match(Scanner.TKT.PUBLIC);
  match(Scanner.TKT.INTERFACE);
  expect(Scanner.TKT.NEWTYPE);
  String N = (String) scan.getTokenval();
  match(Scanner.TKT.NEWTYPE);
  match(Scanner.TKT.OPEN_CURLYBRACKET);
  ArrayList<FdStr> F = new ArrayList<>();
  F = FieldList(F);
  F.get(0).printName();
  match(Scanner.TKT.CLOSE_CURLYBRACKET);
  return new InterStr(A,N,F);
 default: return null;
 }
}
private AnnotationStructure parseAnnotaion(){
 switch (lookahead) {
 case CLOSE_BRACKET: return null;
 default:
  match(Scanner.TKT.AT);
  expect(Scanner.TKT.NAME);
  String name = (String) scan.getTokenval();
  match(Scanner.TKT.NAME);
  match(Scanner.TKT.OPEN_BRACKET);
  Map<String,String> KeyValues = new HashMap<String, String>();
  KeyValues = KeyValuesList(KeyValues);
```

```
   match(Scanner.TKT.CLOSE_BRACKET);
   return new AnnotationStructure(name,KeyValues);
  }
}
private Map<String,String> KeyValuesList(Map<String,String> KeyValues){
 switch (lookahead) {
 case CLOSE_BRACKET:     return null;
 default:
   expect(Scanner.TKT.KEY);
   String K = (String) scan.getTokenval();
   match(Scanner.TKT.KEY);
   match(Scanner.TKT.ASSIGN);
   match(Scanner.TKT.DQUOTE);
   String V;
   switch (lookahead) {
   case NUMBER:
    expect(Scanner.TKT.NUMBER);
    V = (String) scan.getTokenval();
    match(Scanner.TKT.NUMBER);
    break;
   case VALUE:
    expect(Scanner.TKT.VALUE);
    V = (String) scan.getTokenval();
    match(Scanner.TKT.VALUE);
    break;
   default: V= null; break;
   }
   match(Scanner.TKT.DQUOTE);
   KeyValues.put(K, V);
   if (lookahead == Scanner.TKT.COMMA){match(Scanner.TKT.COMMA);
    return KeyValuesList(KeyValues);}
   else return KeyValues;
  }
}
private ArrayList<FdStr> FieldList(ArrayList<FdStr> F){
 switch(lookahead){
 case CLOSE_CURLYBRACKET: return F;
 default:
   FdStr temp = new FdStr(parseField());
   F.add(temp);
   return (lookahead == Scanner.TKT.AT) ? FieldList(F) : F;
  }
}
private FdStr parseField(){
 switch(lookahead){
 case CLOSE_CURLYBRACKET: return null;
 default:
   AnnotationStructure A = new AnnotationStructure(parseAnnotaion());
   ArrayList<Type> T = new ArrayList<>();
   T = TypeList(T);
   expect(Scanner.TKT.NAME);
   String N=(String)scan.getTokenval();
   match(Scanner.TKT.NAME);
   match(Scanner.TKT.SEMICOLON);
   return new FdStr(A,T,N);
  }
}
private ArrayList<Type> TypeList(ArrayList<Type> T){
 Type t = new Type(parseType());
 if(!t.printType().equals("NULL"))      T.add(t);
 switch (lookahead) {
 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("");
  lines.add("public␣class␣" + Inter.printName() + "{");
  ArrayList<String> Constructor = new ArrayList<>();
  String linesConstructor = "public␣" + Inter.printName() + "(";
  Constructor.add(linesConstructor);
  for(FdStr Field : ListofFields){
   ArrayList<Type> type = new ArrayList<Type>(Field.printType());
   String lineField = "protected␣";
   for(int itype=0;itype<type.size();itype++){
   if(type.get(itype).printType().equals("List"))
    lines.set(0, "import␣java.util.List;" + "\nimport␣java.util.ArrayList;");
    if(Scanner.sb.get(type.get(itype).printType().toString()).equals(Scanner.TKT.NEWTYPE) && type.size()==1)
     lines.set(1,"public␣class␣" + Inter.printName() + "␣extends␣" + type.get(itype).printType().toString() +"{");
    lineField += type.get(itype).printType();
    linesConstructor += type.get(itype).printType();
    if(itype+1<type.size()){
     lineField += "<";  linesConstructor+= "<";
    }
   }
   for(int closeBG=0;closeBG<type.size()-1;closeBG++){
    lineField += ">";  linesConstructor += ">";
   }
   lineField += "␣" + Field.printName() + ";";
   lines.add(lineField);
   if (ListofFields.get(ListofFields.size()-1).equals(Field))
    linesConstructor += "␣" + Field.printName() + "_temp";
   else linesConstructor += "␣" + Field.printName() + "_temp,";
    Constructor.add("this."+Field.printName()+"␣="+Field.printName()+"_temp;");
  }
  lines.add("public␣" + Inter.printName() + "(){}");
  linesConstructor += "){";
  Constructor.add("}}"); Constructor.set(0, linesConstructor);
  try {
   file1.createNewFile();  FileWriter writer = new FileWriter(file1);
   for(String f: lines) writer.write(f+"\n");
   for(String f: Constructor) writer.write(f+"\n");
   writer.flush();  writer.close();
  } catch (IOException e) {e.printStackTrace();}
 }
}}
```

Code 8: SQL Generator.

```
public class SQLGenerator extends JavaGenerator{
public ArrayList<String> lines = new ArrayList<>();
public SQLGenerator(ArrayList<InterStr> in) throws IOException {
 super(in);
 String NameFile = "SQLGenerator.sql";
 File file = new File(NameFile); file.createNewFile();
 FileWriter writer = new FileWriter(file);
 for(InterStr Inter : data){
  lines.add("CREATE_TABLE_" + Inter.printAnnotation().getValue("name") + "␣(␣id␣INT_NOT_NULL_PRIMARY_KEY");
  lines.set(lines.size()-1, lines.get(lines.size()-1).split("\\,")[0]);
  lines.add(");\n");
 }
 try {
  for(String f: lines) writer.write(f+"\n");
  writer.flush(); writer.close();
 } catch (IOException e) {e.printStackTrace();}
}
public String printSQL(FdStr f){
 if(f.Type.size()==1){
 switch (f.Type.get(0).printType()) {
```

```
 case "String": String len = f.Annotation.getValue("length");
  if (len.contains(".")) return "VARCHAR("+(len.split("\\."))[0]+")";
  return "VARCHAR(" + len + ")";
 case "Integer": if(f.Annotation.getValue("name").equals("id")) return "INT";
 default:
  if(Scanner.sb.get(f.Type.get(0).printType()).equals(Scanner.TKT.NEWTYPE) && f.
      Annotation.getValue("target").equals(f.Type.get(0).printType())){
   String name = f.Annotation.getValue("name");
   String target = f.Annotation.getValue("target");
   String out = "INT," + "ADD␣FOREIGN␣KEY␣(" + name + ")␣REFERENCES␣`" + target.
       toLowerCase() + "`(`id`)";
   return out;
  } return null;
 }
}else return null;
}
public void printNewType(){
 String NameFile = "SQLGenerator.sql";
 for(InterStr Inter : data){
  ArrayList<FdStr> ListofFields = new ArrayList<>(Inter.printField());
  lines.add("ALTER␣TABLE␣`" + Inter.printAnnotation().getValue("name") + "`␣");
  for(int ifield=1; ifield<ListofFields.size();ifield++){
   String lineField = null;
   if (printSQL(ListofFields.get(ifield))!= null){
    lineField = "ADD␣COLUMN␣" + ListofFields.get(ifield).printAnnotaion().
        getValue("name") + "␣";
    lineField += printSQL(ListofFields.get(ifield));
    if(ifield+1<ListofFields.size()) lineField += ",";
    lines.add(lineField);
   }else lines.set(lines.size()-1,lines.get(lines.size()-1).split("\\,")[0]);
  }
  lines.add(";\n");
 }
 try {
  File file = new File(NameFile); file.createNewFile();
  FileWriter writer = new FileWriter(file);
  for(String f: lines) writer.write(f+"\n");
  writer.flush(); writer.close();
 } catch (IOException e) {e.printStackTrace();}
}}
```

0.4 Exercise 4.

Code 9: IEntityManger Class.

```
public class IEntityManagerClass<T> implements IEntityManager<T>{
protected Class<T> type;
protected int waitPublisher = 0;
public IEntityManagerClass(Class<?> tem){this.type = (Class<T>) tem;}
public IEntityManagerClass(){}
@Override
public void persist(T entity) {
 File file = new File("SQLGenerator.sql");
 StringBuilder sb = new StringBuilder();
 Class<?> thisClass = null;
 try {
  file.createNewFile(); FileWriter writer = new FileWriter(file,true);
  thisClass = Class.forName(entity.getClass().getName());
  java.lang.reflect.Field[] aClassFields = thisClass.getDeclaredFields();
  sb.append("INSERT␣INTO␣" + entity.getClass().getSimpleName() + "␣VALUES(");
  for(java.lang.reflect.Field f : aClassFields){
   if(f.get(entity)!= null){
    if(f.getType().getSimpleName().equals("String")){
     if(f==aClassFields[aClassFields.length-1]) sb.append(f.get(entity));
     else sb.append("\'" + f.get(entity) + "\'" + ",␣");
```

```java
    }else if(f.getType().getSimpleName().equals("Integer")){
     if(f==aClassFields[aClassFields.length-1]) sb.append(f.get(entity));
     else sb.append(f.get(entity) + ",␣");
    }else{
      java.lang.reflect.Field[] aClassFields2 = f.getType().getDeclaredFields();
      for(java.lang.reflect.Field f2 : aClassFields2)
       if(f2.getName().equals("id")) sb.append(f2.get(f.get(entity)));
    }
   }else sb.append("NULL");
  }
  sb.append(");");
  writer.write("\n" + sb.toString());  writer.flush();  writer.close();
 } catch (Exception e) {e.printStackTrace();}
}
@Override
public void remove(T entity) {
 StringBuilder sb = new StringBuilder();
 Class<?> thisClass = null;
 File file = new File("SQLGenerator.sql");
 try {
  file.createNewFile(); FileWriter writer = new FileWriter(file,true);
  thisClass = Class.forName(entity.getClass().getName());
  java.lang.reflect.Field[] aClassFields = thisClass.getDeclaredFields();
  sb.append("DELETE␣FROM␣" + entity.getClass().getSimpleName().toLowerCase() + "␣WHERE␣");
  for(java.lang.reflect.Field f : aClassFields)
   if(f.get(entity)!= null && f.getName().equals("id"))
    sb.append(f.getName() + "␣=␣" + f.get(entity));
   else continue;
  sb.append(";");
  writer.write("\n" + sb.toString());  writer.flush();  writer.close();
} catch (Exception e) {e.printStackTrace();}
}
@Override
public T find(Object pk) {
 T out = null;
 try {Connection con = DriverManager.getConnection(url,username,password);
  out = type.newInstance();
  String q = ("SELECT␣*␣FROM␣" + type.getName().toLowerCase() + "␣WHERE␣id␣=␣" + pk + ";");
  java.sql.PreparedStatement st = con.prepareStatement(q);
  ResultSet result = st.executeQuery();
  ResultSetMetaData metaData = result.getMetaData();
  if(result.next()){
   java.lang.reflect.Field[] ListFields = type.getDeclaredFields();
   int specialPoision = -1;  int icol = 1;
   while(icol<=metaData.getColumnCount()){
    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")))
     ListFields[icol-1].set(out,result.getObject(icol));
    else if(metaData.getColumnTypeName(icol).toString().toUpperCase().equals("INT") && waitPublisher == 0){
     specialPoision = icol;
     Class<?> tem = ListFields[icol-1].getType();
     IEntityManagerClass<T> a = new IEntityManagerClass<T>(tem);
     ListFields[icol-1].set(out,a.find(result.getObject(icol)));
    }else if(metaData.getColumnTypeName(icol).toString().toUpperCase().equals("INT") && waitPublisher == 1)
      ListFields[icol-1].set(out,null);
     else specialPoision = icol;
   icol++;
   }
   if(specialPoision == -1)     specialPoision = icol -1;
   if(metaData.getColumnCount()<ListFields.length){
    String[] Types_ = ListFields[specialPoision].getGenericType().toString().
        split("\\W");
    String TempClass = Types_[Types_.length -1];
    Class<?> tem2 = Class.forName(TempClass);
    java.lang.reflect.Field[] TempClassFields = tem2.getDeclaredFields();
    String que ="select␣*␣from␣"+tem2.getSimpleName().toLowerCase()+"␣where␣"+
        tem2.getSimpleName().toLowerCase()+"."+TempClassFields[specialPoision].
        getName()+"␣=␣"+pk+";";
    java.sql.PreparedStatement secConnec = con.prepareStatement(que);
    ResultSet secResult = secConnec.executeQuery();
    List<T> ListBook = new ArrayList<>();
    while(secResult.next()){
     IEntityManagerClass<T> retrBook = new IEntityManagerClass<T>(tem2);
     retrBook.waitPublisher = 1;
     ListBook.add(retrBook.find(secResult.getObject(1)));
    }
    ListFields[icol -1].set(out,ListBook);
    for(int iT=0; iT<ListBook.size();iT++){
     java.lang.reflect.Field[] ListField_elemBook = tem2.getDeclaredFields();
     for(java.lang.reflect.Field f : ListField_elemBook)
      if(f.getType().getSimpleName().toString().equals(type.getName())) f.set(
          ListBook.get(iT),out);
    }
    ListFields[icol -1].set(out,ListBook);
   }
  }else return null;
 } catch (Exception e) {e.printStackTrace();}
return (T) out;
}
@Override
public Query<T> createQuery(String query) {
 Query<T> out = new Query<>(query, type);
 return out;
}}
```

0.5 Exercise 5.

Code 10: Query Class.

```
public class Query<T> implements IQuery<T> {
private Class<T> typeOfClass;
private String query;
protected Query(String q,Class<T> A) {
 this.query = q; this.typeOfClass = A;
}
@Override
public List<T> getResultList() {
 List<T> out = new ArrayList<T>();
 IEntityManagerClass<T> retr = new IEntityManagerClass<T>(typeOfClass);
 try {Connection con = DriverManager.getConnection(url,username,password);
  String q = ("SELECT␣*␣FROM␣" + typeOfClass.getName().toLowerCase() + ";");
  java.sql.PreparedStatement st = con.prepareStatement(q);
  ResultSet result = st.executeQuery();
  while(result.next()) out.add(retr.find(result.getObject(1)));
 } catch (SQLException e) {e.printStackTrace();}
 return out;
}
@Override
public void execute() {
 try{Connection con = DriverManager.getConnection(url,username,password);
  java.sql.PreparedStatement st = con.prepareStatement(query);
  st.execute();
} catch (Exception e) {e.printStackTrace();}
}}
```

0.6 Exercise 6.

Object-Relational Mapping (ORM) is a programming technique for converting or transforming data between Object-Oriented programming language and incompatible type systems based on a "virtual object database". There are two packages free and commercial [1]. That means it can hide the SQL in the code and the database can be easily accessed. Therefore, instead of accessing directly SQL server to retrieve a query and process data, ORM interacts through programming languages. Obviously, it is more readable and fewer errors (reducing the amount of code and make the software more robust). The example below shows the interaction of ORM in Java.

Code 11: Example of retrieving data from mySQL by Java.

```
String query = "SELECT * FROM book WHERE id = 10";
java.sql.PreparedStatement st = connection.prepareStatement(query);
ResultSet result = st.executeQuery();
String title = result.next().getString("title");
```

Code 12: Example of retrieving data based on ORM.

```
Book b = IEntityManagerClass.find(10);
String title = b.gettitle();
```

The Java Persistence API (JPA) being a specification for the persistence of Java objects is designed by Sun Microsystems. this API requires J2SE 1.5 ($\geqslant$ Java5), as it makes heavy use in features of Java programming language including annotations and generics [3]. For instance, with retrieving information from table Magazine, the code below can illustrate the way of the API.
EntityManager em = ...
Query q = em.createQuery("SELECT x FROM Magazine x");
List$< Magazine >$ results = (List$< Magazine >$) q.getResultList();

The = operator tests for equality. <> tests for inequality. JPQL also supports the following arithmetic operators for numeric comparisons: $>, >=, <, <=$. The AND, OR and NOT logical operators chain multiple criteria together [3]:
AND: SELECT x FROM Magazine x WHERE x.p > 3 AND x.p <5
OR : SELECT x FROM Magazine x WHERE x.t = 'a' OR x.t = 'b'
NOT: SELECT x FROM Magazine x WHERE NOT(x.p = 10)

LINQ simplifies the expression (retrieving data) by offering a consistent model. Thus, it support working with data across various types and formats. In the query, Linq works with objects and requires basic code to query and covert data from XML, SQL or ADO.NET and any other [4]. For example, the logical operators of Linq in C# is represented below [5].
AND (&&): var links = links.Where(l ⇒ l.First()=='/' && l.First()=='//').ToList();
OR (||): var links = links.Where(l ⇒ l.First()=='/' || l.First()=='//').ToList();
NOT (!) : var query = from item in context.items where !ids.Contains(item.id) select item;

After these examples, it can be said that the query of JPA from Java is more similar to the SQL shema query than queries of Linq from C#. However, the Linq is utilized the anonymous function or lambda expression and close to the programming language (for instance, using some symbols such as && and ||).

0.7 Reference

1. https://en.wikipedia.org/wiki/Object-relational_mapping - Access 27 Sep 2017
2. https://stackoverflow.com/questions/1152299/ - Access 27 Sep 2017
3. https://openjpa.apache.org/builds/1.0.1/apache-openjpa-1.0.1/docs/manual/jpa_overview_query.html - Access 27 Sep 2017
4. https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries - Access 27 Sep 2017
5. https://stackoverflow.com/questions/18765161/using-or-in-linq-expressions - Access 27 Sep 2017