Forma de entrega: fazer a atividade descrita a seguir e mostrar para o professor na aula do dia 04/maio (sexta-feira). A entrega pode ser em grupos de até 4 pessoas.

A entrega deverá contemplar o código, diagramas UML de classes e de Sequência e testes unitários.

Descrição da atividade: fazer uma aplicação que o usuário fornece o nome de uma cidade e o sistema retorna para ele a previsão do tempo para os próximos 7 dias. A previsão deverá ser formada por temperatura mínima, temperatura máxima e IUV (Índice Ultravioleta).

O CPTEC-INPE possui serviços de previsão do tempo que podem ser consumidos usando apenas uma URL, a lista pode ser visualizada em http://servicos.cptec.inpe.br/XML/. Dentre as opções existe a previsão para 7 dias que está disponível pela seguinte URL:

http://servicos.cptec.inpe.br/XML/cidade/7dias/4963/previsao.xml

O número 4963 é o código da cidade de São José dos Campos. O código de cada cidade pode ser obtido usando a seguinte URL, onde o nome do município pode ser parcial:

http://servicos.cptec.inpe.br/XML/listaCidades?city=sao%20jose

Requisitos do sistema:

1. O banco de dados deverá ser implementado no SQLite e deverá ter as tabelas ilustradas na Figura 1;
2. O sistema deverá manter os dados das cidades já consultadas na tbcidade para evitar ter de buscar no serviço do CPTEC o código da cidade a cada nova consulta da mesma cidade;
3. O sistema deverá manter os dados de previsão já consultados na tbprevisao e a data de atualização no campo tbcidade.atualizacao. Se no momento da consulta a data de atualização da previsão for diferente de hoje, todos os dados de previsão da cidade deverão ser removidos da tbprevisao e novos dados deverão ser buscados no serviço do CPTEC;
4. O sistema deverá ter uma interface gráfica ou por linha de comando para o usuário fornecer o nome de uma cidade ou parte do nome. Na sequência o sistema deverá exibir como resultado a previsão do tempo para os próximos 7 dias - formada por data (dd/mm/aaaa), tempo, IUV, temperatura mínima e máxima. No caso do nome fornecido pelo usuário resultar em vários nomes, por exemplo, São José, o sistema deverá exibir somente o 1º resultado da consulta.

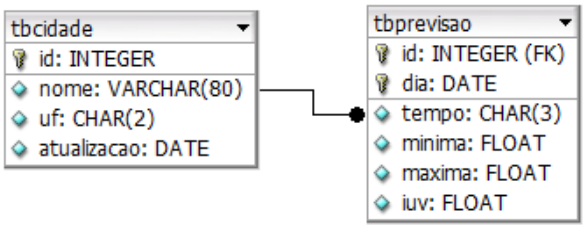

Figura 1 – Modelo do BD.

Dicas: essas dicas não necessariamente precisam ser seguidas.

a. Acesse https://bitbucket.org/xerial/sqlite-jdbc/downloads/ para obter o driver do SQLite. Para instalar o driver basta colocar o arquivo .jar na estrutura do projeto, assim como ilustrado na Figura 2. O procedimento para incluir esse JAR no classpath do projeto é igual ao procedimento para incluir a biblioteca Mockito.

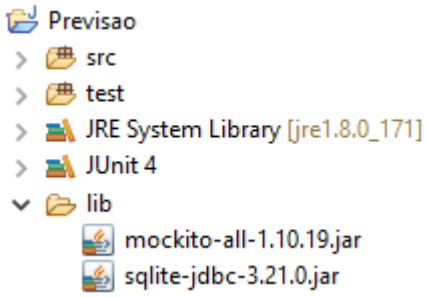

Figura 2 – Estrutura do projeto no Eclipse. O processo de importar o SQLite é igual ao do Mockito.

b. A 1ª vez que o programa for executado o BD e suas tabelas deverão ser criadas. A Figura 3 e Figura 4 mostram exemplos de códigos para efetuar a conexão com o BD, criar o BD e a tabela. O arquivo bdprevisao.db será criado na raiz do diretório do seu projeto.

```
public Connection conectar() throws SQLException, ClassNotFoundException {
      Connection conexao = null;
      Class.forName("org.sqlite.JDBC");

      File bd = new File("bdprevisao.db");
      /* verifica se o arquivo do BD existe na raiz do projeto */
      if( !bd.exists() ){
         /* cria o arquivo do BD na raiz do projeto e cria uma conexão para o BD */
         conexao = DriverManager.getConnection("jdbc:sqlite:bdprevisao.db");
         /* como o BD não existe então é necessário criar as tabelas */
         createTableCidade();
         createTablePrevisao();
      }
      else{
         /* cria uma conexão com o BD */
         conexao = DriverManager.getConnection("jdbc:sqlite:bdprevisao.db");
      }
      conexao.setAutoCommit(false);
      return conexao;
   }
```

Figura 3 – Método para criar o BD e retornar uma conexão para o BD.

```
public boolean createTablePrevisao() throws SQLException{
   Statement stmt = conexao.createStatement();
   String sql = "create table if not exists tbprevisao( " +
                "id int not null," +
                "dia date not null," +
                "tempo char(3) not null," +
                "minima float not null," +
                "maxima float not null," +
                "iuv float not null," +
                "primary key (id, dia)," +
                "foreign key (id) references tbcidade(id) " +
               ")";
   stmt.executeUpdate(sql);
```

```
    stmt.close();
    return true;
}
```

Figura 4 – Método para criar a tbprevisao no BD.

c. A Figura 5 mostra um exemplo de código para inserir um registro na tbcidade. É boa prática usar prepared statement para evitar SQL injection. A Figura 6 mostra um exemplo para retornar os registros da tbcidade.

```
public boolean insertCidade(Cidade cidade) throws SQLException{
    /* o campo atualizacao irá receber o valor padrão, ou seja, null */
    String sql = "insert or ignore into tbcidade(id,nome,uf) values(?,?,?)";
    PreparedStatement stmt = conexao.prepareStatement(sql);
    stmt.setInt(1, cidade.getId() );
    stmt.setString(2, cidade.getNome() );
    stmt.setString(3, cidade.getUf() );
    stmt.execute();
    stmt.close();
    conexao.commit();
    return true;
}
```

Figura 5 – Método para inserir um registro na tbcidade.

```
public List<Cidade> selectCidade(String sql) throws SQLException{
    Statement stmt = conexao.createStatement();
    ResultSet rs = stmt.executeQuery(sql);
    List<Cidade> lista = new ArrayList<>();
    Cidade cidade;
    while ( rs.next() ) {
        cidade = new Cidade();
        cidade.setId(rs.getInt("id"));
        cidade.setNome(rs.getString("nome"));
        cidade.setUf(rs.getString("uf"));
        cidade.setAtualizacao(rs.getString("atualizacao"));
        lista.add(cidade);
    }
    rs.close();
    stmt.close();
    conexao.commit();
    return lista;
}
```

Figura 6 – Método para retornar os registros da tbcidade.

d. A Figura 7 mostra um exemplo de código para fazer uma conexão com o serviço do CPTEC e obter os dados de uma cidade. Já o código da Figura 8 mostra um exemplo para converter a string no formato XML em um objeto Java. As classes da Figura 9 são necessárias na interpretação das marcações XML.

```
public String getXMLCidade(String cidade) throws Exception {
    String charset = java.nio.charset.StandardCharsets.ISO_8859_1.name();
    String linha, resultado = "";
    String urlListaCidade = "http://servicos.cptec.inpe.br/XML/listaCidades?city=%s";
    /* codifica os parâmetros */
    String parametro = String.format(urlListaCidade, URLEncoder.encode(cidade, charset) );
    URL url = new URL(parametro);
    URLConnection conexao = url.openConnection();
```

```
   BufferedReader reader = new BufferedReader(new InputStreamReader(conexao.getInputStream()));
   while((linha = reader.readLine()) != null){
      resultado += linha;
   }

   return resultado;
}
```

Figura 7 – Método para obter no serviço do CPTEC os dados de uma cidade.

```
public Cidade[] xmlToObjectCidade(String xml) throws Exception {
   StringReader sr = new StringReader(xml);
   /* a base do XML é uma marcação de nome cidades */
   JAXBContext context = JAXBContext.newInstance(Cidades.class);
   Unmarshaller un = context.createUnmarshaller();
   Cidades cidades = (Cidades) un.unmarshal(sr);
   return cidades.getCidade();
}
```

Figura 8 – Método para transformar uma string no formato XML para um objeto Java.

```
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "cidade")
@XmlType(propOrder = {"nome", "uf", "id"})
public class Cidade {
   @XmlElement(name = "id")
   private Integer id;
   @XmlElement(name = "nome")
   private String nome;
   @XmlElement(name = "uf")
   private String uf;

}
```

```
@XmlRootElement(name = "cidades")
@XmlType(propOrder = {"cidade"})
public class Cidades {
   @XmlElement
   private Cidade[] cidade;

}
```

Figura 9 – Parte das classes usadas para representar marcações XML.