Deploying and running JPA application client

The Java Persistence API (JPA) is a new programming model under EJB3.0 specification (JSR220) for the management of persistence and object/relational mapping with Java EE and Java SE. With JPA, developers can easily develop java applications that perform operations on relational database management systems using java objects and mapping. In that way, java applications developed using JPA are not only portable across different platforms, but also applications can be easily developed using simple yet powerful programming model provided by JPA. This greatly improves application maintainability against ever changing database world. JPA insulates applications from all the complexity and non-portable boilerplate code involved in database connectivity and operations.

Apache Geronimo uses Apache OpenJPA for providing Java Persistence API implementation to Java EE applications deployed in the server. Even though JPA is a part of EJB3 spec, it is independent of it. Hence, JPA can be used in JavaSE, web and ejb applications in the same uniform way. Think of a JPA implementation, e.g. Apache OpenJPA as a driver for JPA (JPA driver) similarly to what a JDBC driver is for JDBC.

The below section illustrates developing and deploying a JEE application client that uses JPA to access and manipulate database data residing in an embedded derby database.

In order to develop, deploy and run the application, the following environment is required.

- Sun JDK 5.0+ (J2SE 1.5)
- Eclipse 3.3.1.1 (Eclipse Classic package of Europa distribution), which is platform specific
- Web Tools Platform (WTP) 2.0.1
- Data Tools Platform (DTP) 1.5.1
- Eclipse Modeling Framework (EMF) 2.3.1
- Graphical Editing Framework (GEF) 3.3.1

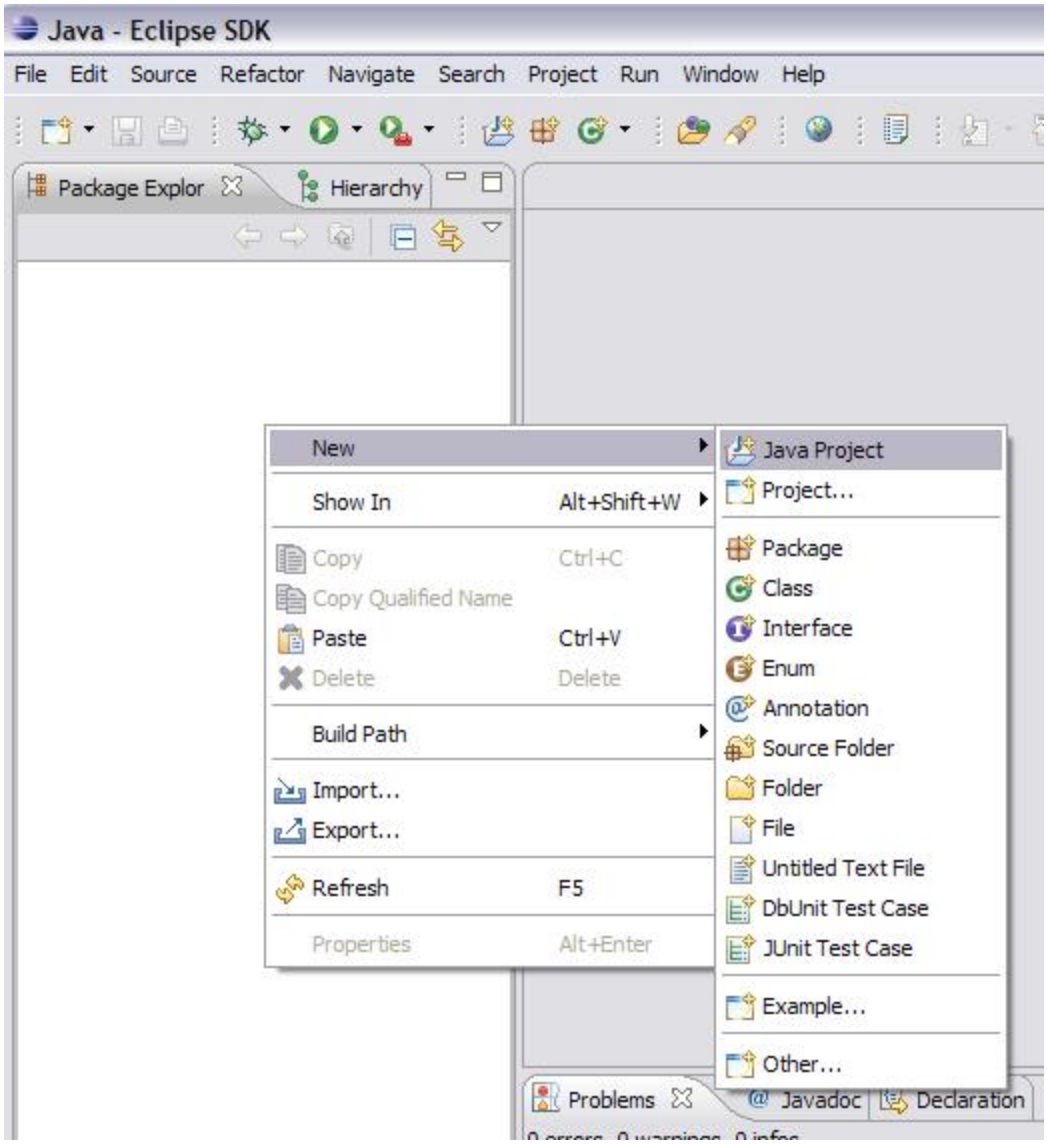
The article has the following sections.

- Setting up Eclipse for Application development
- Developing Entities and Client
- Preparing Deployment Descriptors and Deployment Plans
- Deploying the application client
- Running the application client

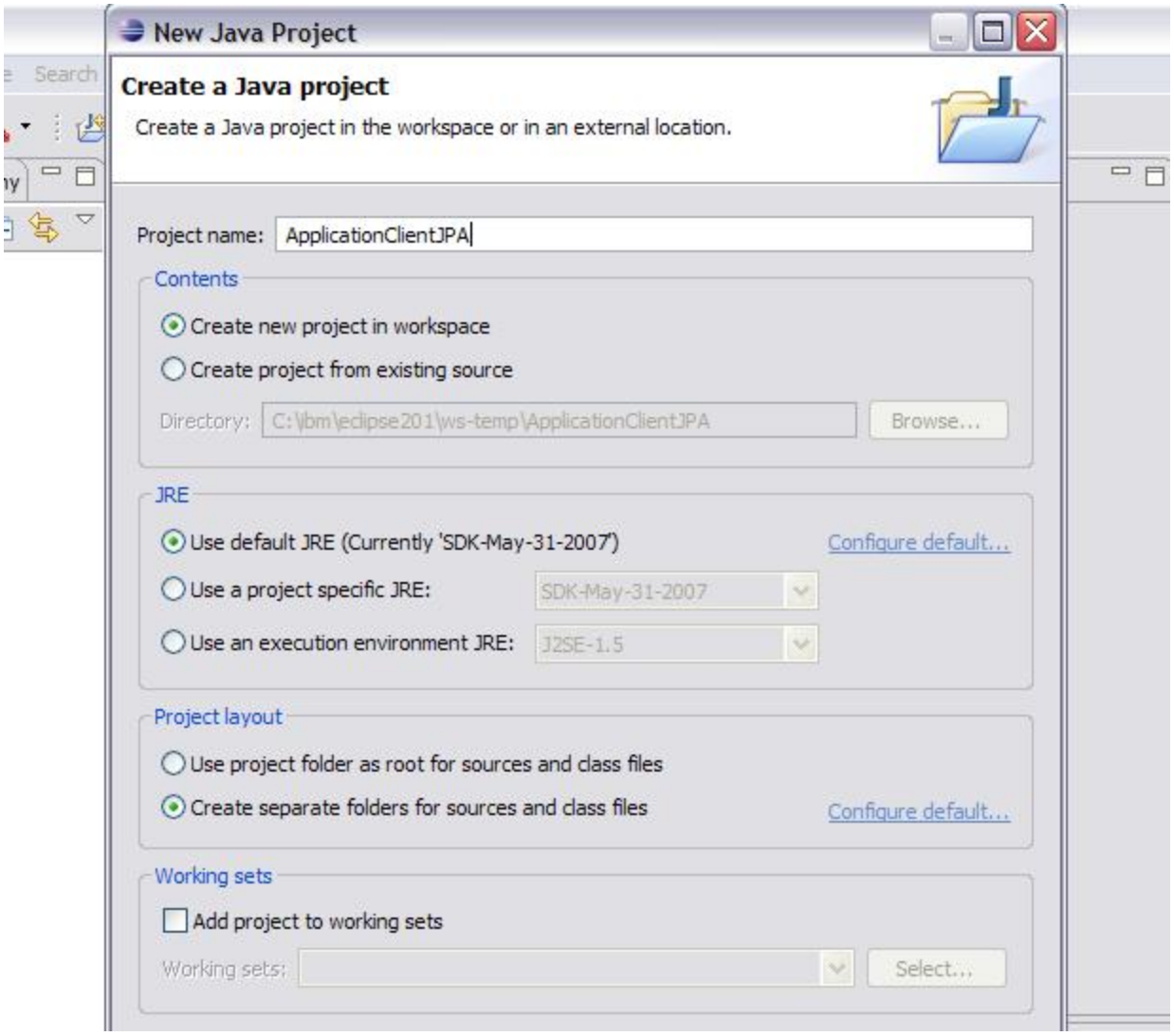
To download the complete application, click on link

Setting Eclipse for Application development

1. Start the Eclipse IDE and create `ApplicationClientJPA` java project.
Right click on the *Project Explorer* Window and click on *New* > *Java Project*.

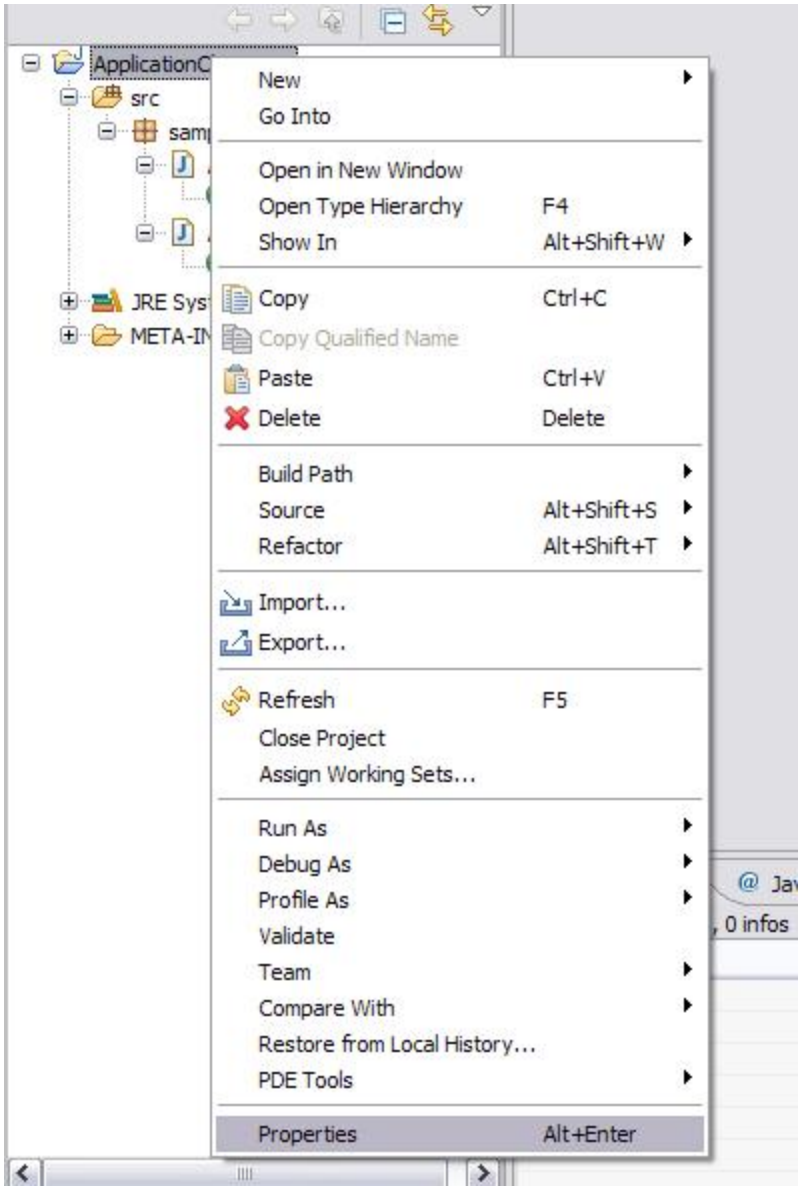


Provide `ApplicationClientJPA` as the name for the Java Project.

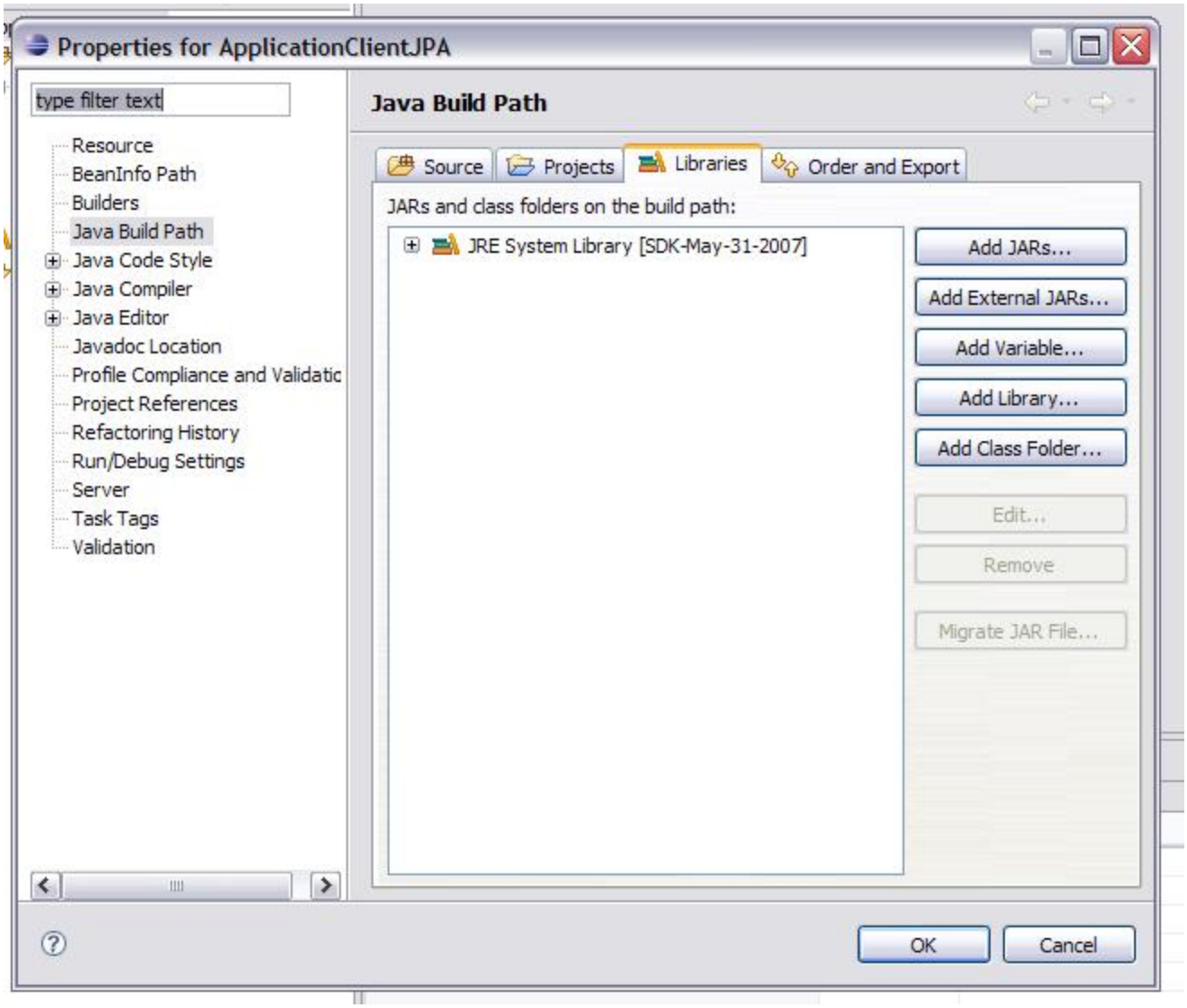


Click on *Finish*.

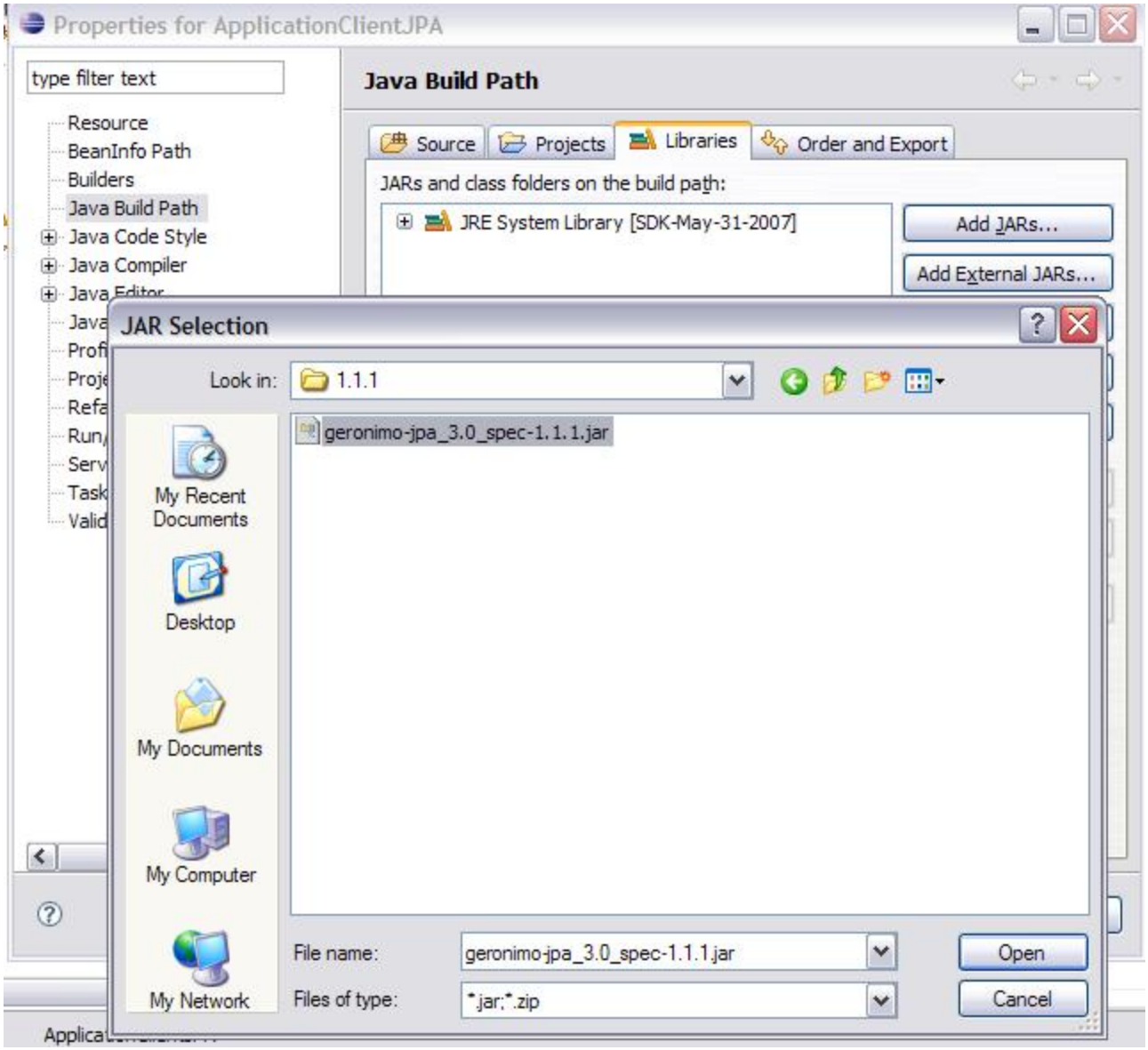
Right click on the Project and click on the *Properties* option which is the last option in the menu.



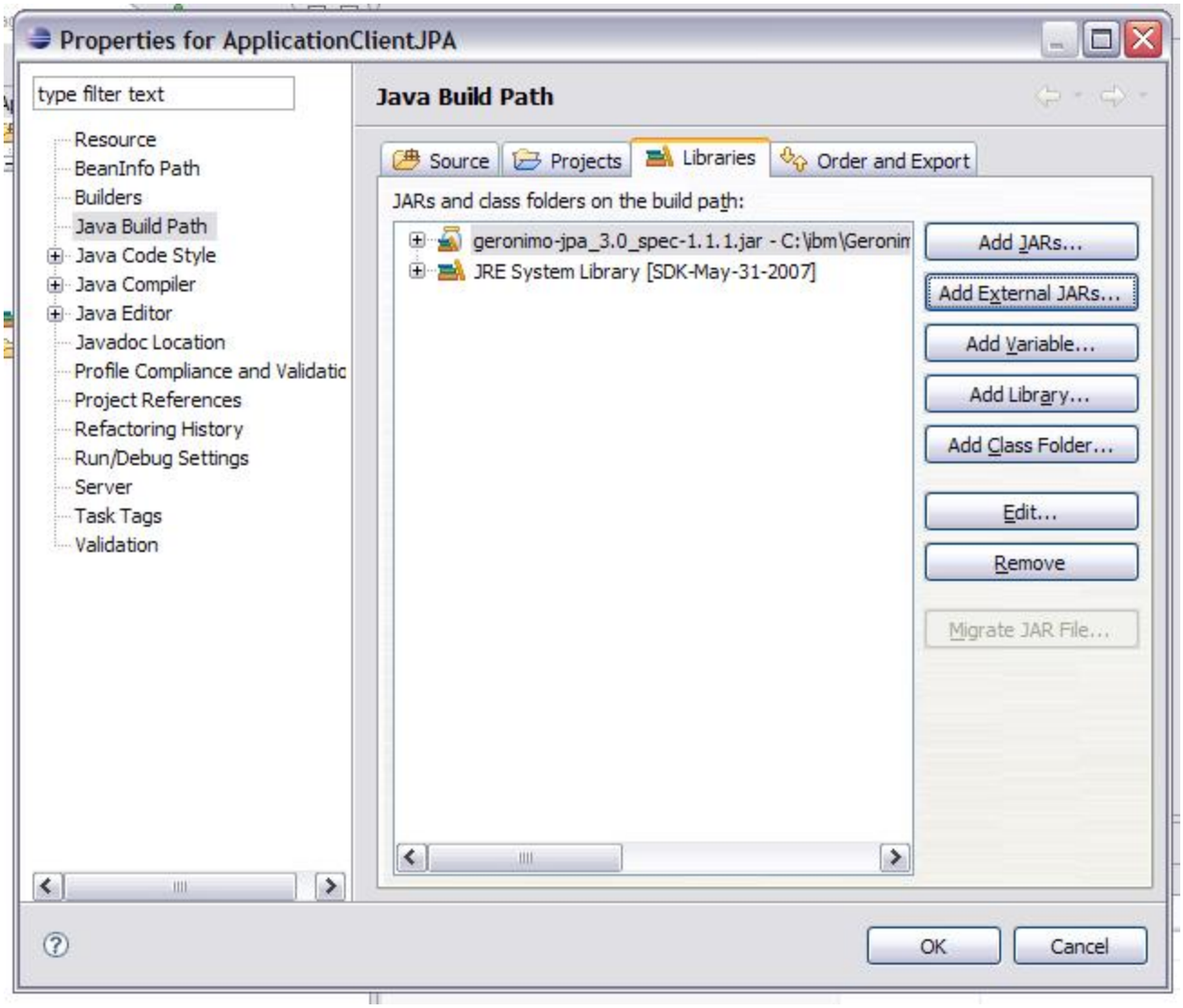
This will open up *Properties* Window. Click on *Java Build Path => Libraries* tab on the right hand side.



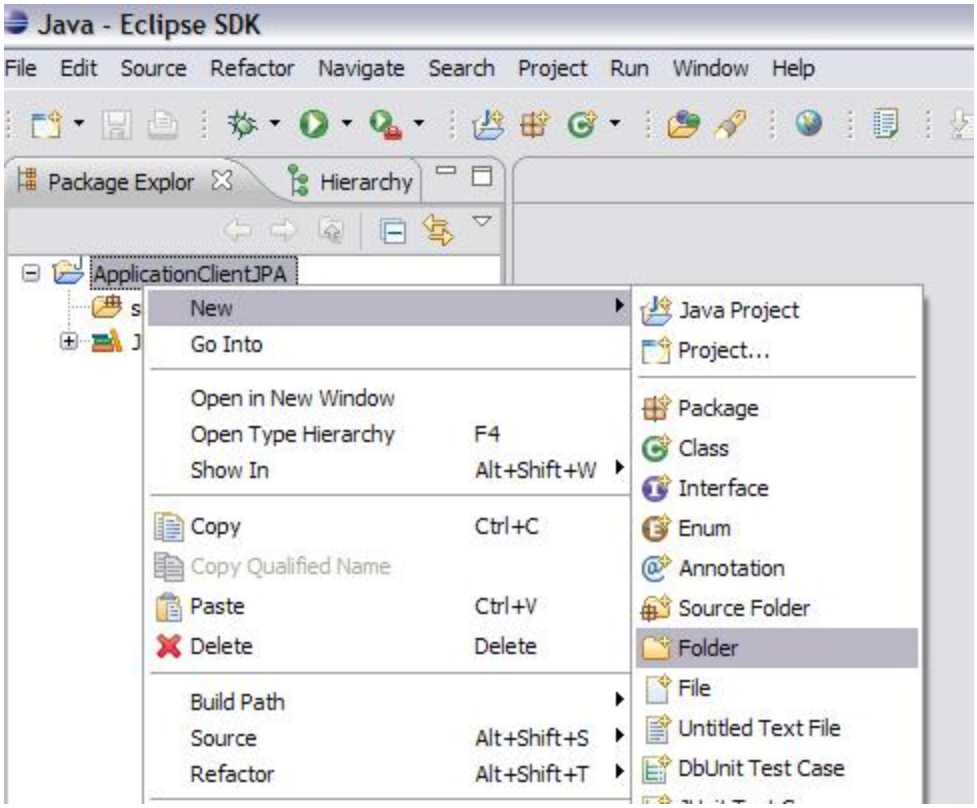
Click on *Add External JARs* button and browse for `geronimo-jpa_3.0_spec-1.1.1.jar` file which should be in `<geronimo_home>/repository/org/apache/geronimo/specs/geronimo-jpa_3.0_spec/1.1.1`. The jar file is only to compile entity and entity client classes you're about to create. Geronimo provides the jar at runtime.
Select the file and click on the *Open* button.



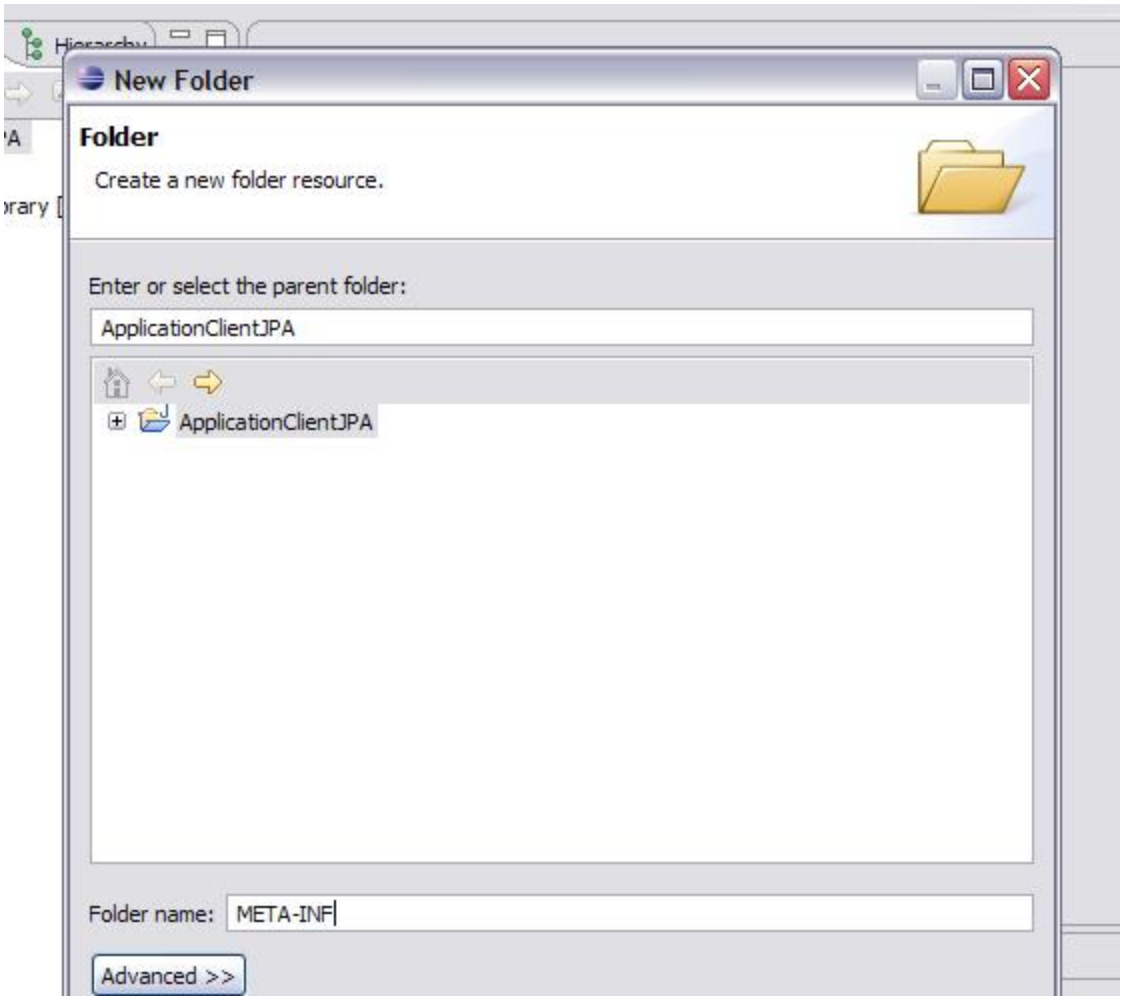
This will add the library to the build path. Click on *OK*.



2. Create `META-INF` folder in the project. Right click on the project and click on *New* > *Folder*



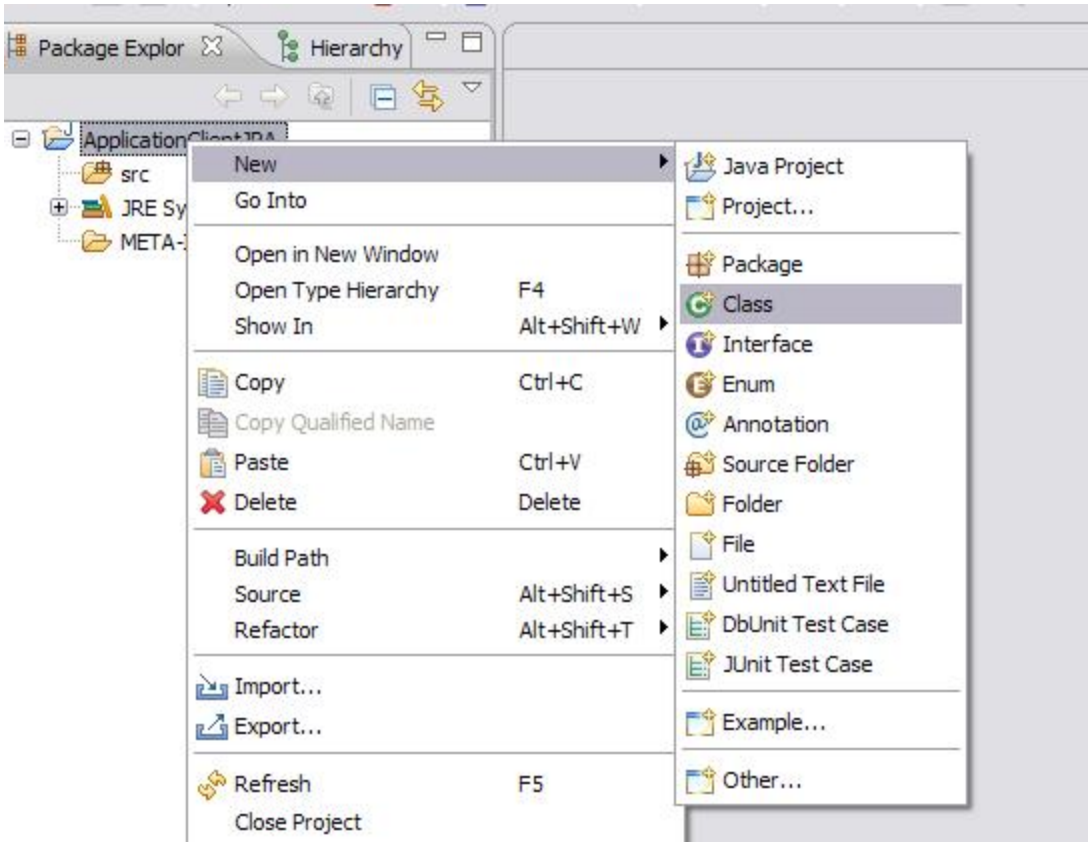
Provide `META-INF` for the folder name and click on *Finish*.



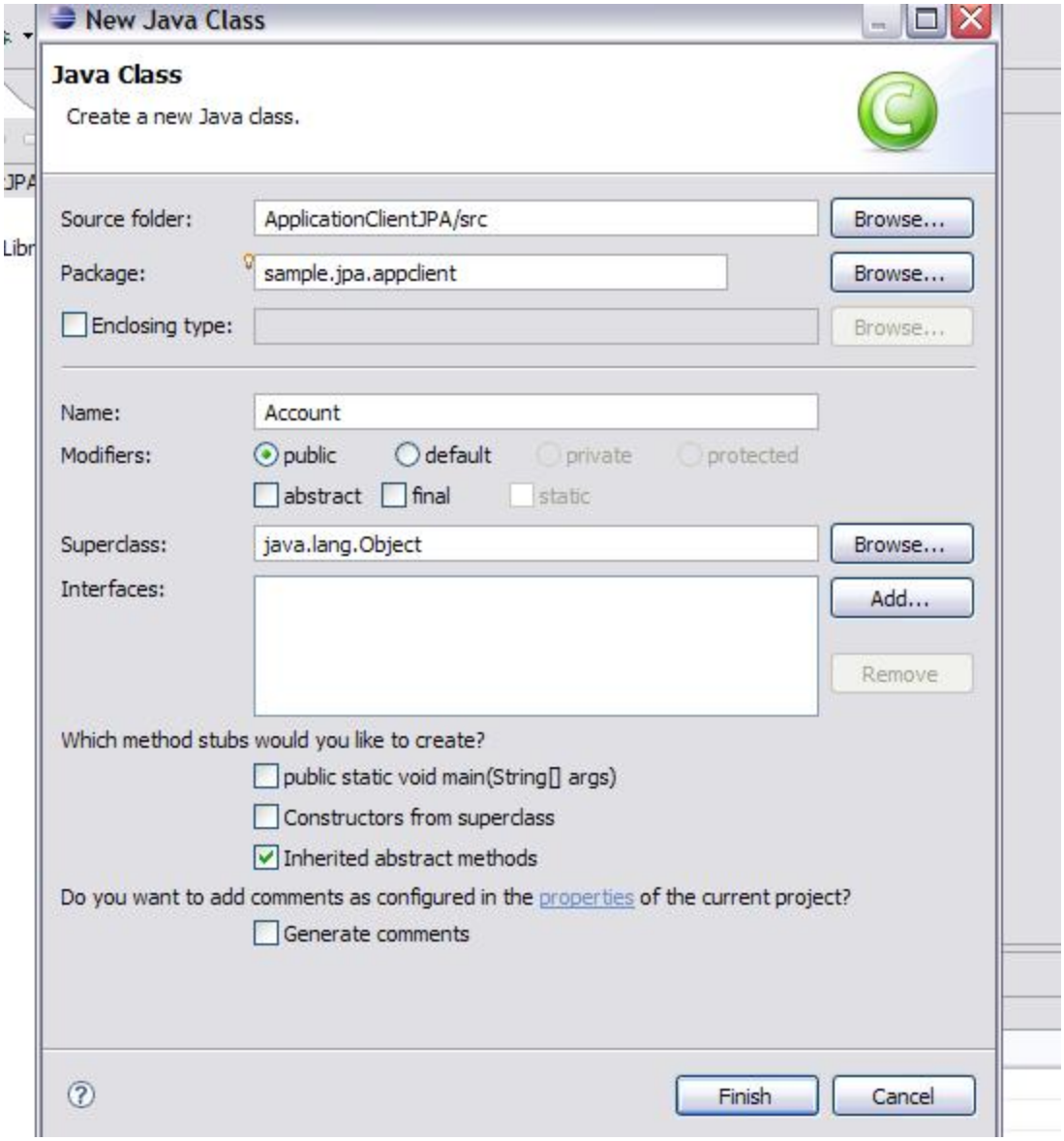
That's all! the setup required for developing the application client is over.

Developing Entities and the Client

1.Right click on the project and create a `Account` java class as follows.



Provide name for the class as `Account` with a package name as `sample.jpa.appclient`. Click on *Finish*.



2. Copy the below contents to the `Account.java`.

Account.java

```
package sample.jpa.appclient;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="Account1")
public class Account implements Serializable{

 private static final long serialVersionUID = 1L;

 @Id
```

```
public int accountNo;
public String name;
public String address;
public int branchCode;
public double balance;

public Account(){
 this.accountNo = 0;
 this.name = "DUMMY";
 this.address = "DUMMY";
 this.branchCode = 0;
}

public int getAccountNo() {
 return accountNo;
}

public void setAccountNo(int accountNo) {
 this.accountNo = accountNo;
}

public String getName() {
 return name;
}

public void setName(String name) {
 this.name = name;
}

public String getAddress() {
 return address;
}

public void setAddress(String address) {
 this.address = address;
}

public int getBranchCode() {
 return branchCode;
}

public void setBranchCode(int branchCode) {
 this.branchCode = branchCode;
}

public double getBalance() {
 return balance;
}

public void setBalance(double balance) {
 this.balance = balance;
}
```

```
}
```

3. Similarly, right click on the project and create another java class by name `AccountClient.java`. Also, provide the same package name `sample.jpa.appclient`.

4.Copy the below contents to the `AccountClient.java`.

AccountClient.java

```
package sample.jpa.appclient;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;
import javax.persistence.EntityTransaction;

public class AccountClient {

    private EntityManager em;

    public AccountClient() {

        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("JPA-App-Client");
        if (emf == null) {
            throw new IllegalStateException("EntityManagerFactory is unavailable");
        }

        em = emf.createEntityManager();
        if (em == null) {
            throw new IllegalStateException("EntityManager is unavailable");
        }
    }

    public static void main(String[] args) {

        AccountClient client = new AccountClient();

        String opt = args[0];
        if ("create".equals(opt)) {
            if (args.length != 6) {
                System.err.println("No values for accountNo, name, address, branchCode
and balance. Exiting.");
                System.exit(0);
            } else {
                int accNo = Integer.parseInt(args[1]);
                String name = args[2];
                String address = args[3];
                int branchCode = Integer.parseInt(args[4]);
                double balance = Double.parseDouble(args[5]);
                client.createAccount(accNo, name, address, branchCode, balance);
            }
```

```
        } else if ("list".equals(opt)) {
            List<Account> accounts = client.listAccounts();
            for (Account account: accounts) {
                System.out.println("_____________________________________");
                System.out.println(account.getAccountNo());
                System.out.println(account.getName());
                System.out.println(account.getAddress());
                System.out.println(account.getBranchCode());
                System.out.println(account.getBalance());
                System.out.println("_____________________________________________________");
                System.out.println("");
            }
        } else if ("update".equals(opt)) {
            if (args.length != 3) {
                System.out.println("No values for accountNo and new balace value.
Exiting.");
                System.exit(0);
            } else {
                int accNo = Integer.parseInt(args[1]);
                double newbalance = Double.parseDouble(args[2]);
                client.updateAccountBalance(accNo, newbalance);
            }
        } else if (opt.equals("delete")) {
            if (args.length != 2) {
                System.out.println("No values for accountNo for delete. Exiting.");
                System.exit(0);
            } else {
                int accNo = Integer.parseInt(args[1]);
                client.deleteAccount(accNo);
            }
        }
        else {
            System.err.println("Unknown option (" + opt + ") selected");
        }

    }

    public Account createAccount(int accNo, String name, String address, int
branchCode, double balance) {

        Account acc1 = em.find(Account.class, accNo);
        if (acc1 != null) {
            throw new IllegalArgumentException("Account already exists - account
Number (" + accNo + ")");
        }
        Account acc = new Account();
        acc.setAccountNo(accNo);
        acc.setAddress(address);
        acc.setBalance(balance);
        acc.setBranchCode(branchCode);
        acc.setName(name);
        System.out.println("Persisting account entity (accNo = " + accNo + ")");
        EntityTransaction et = em.getTransaction();
        et.begin();
        em.persist(acc);
        et.commit();
        System.out.println("Persisted successfully account entity (accNo = " + accNo +
")");
        return acc;
```

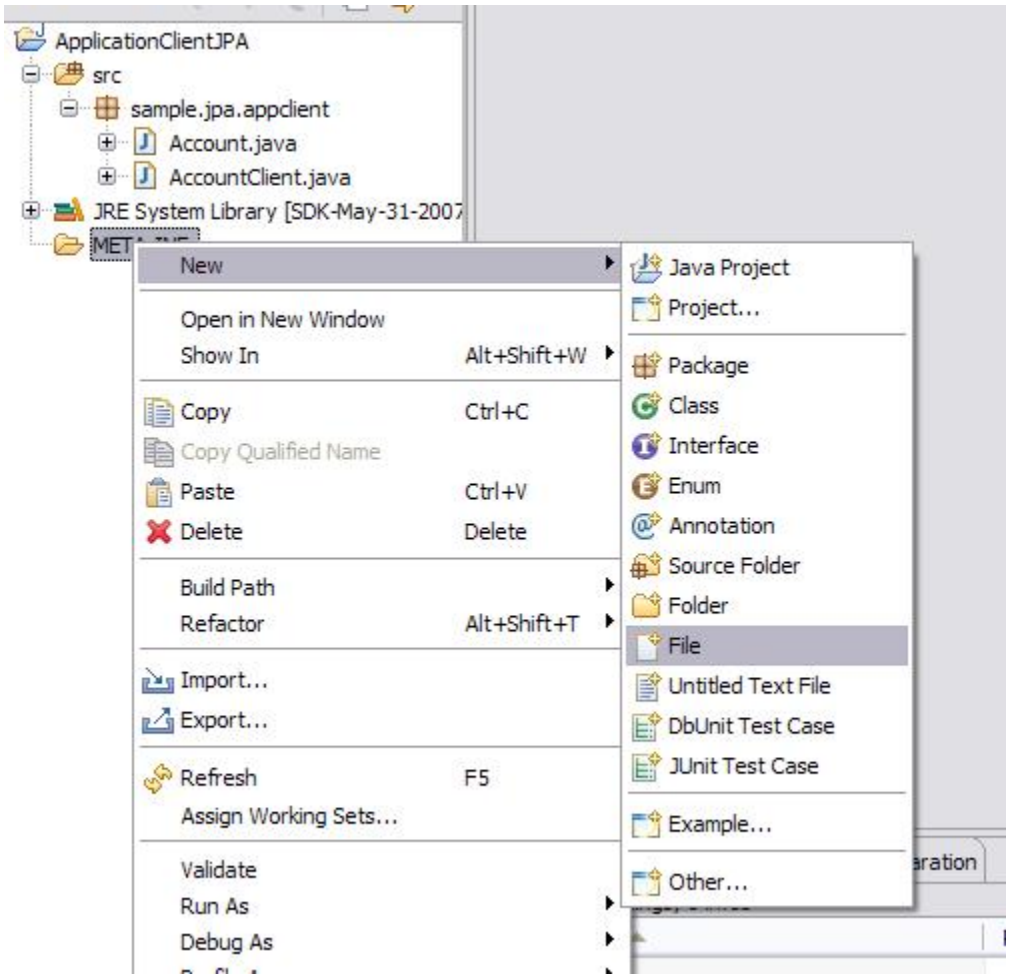
```
    }

    @SuppressWarnings("unchecked")
    public List<Account> listAccounts() {
        Query q = em.createQuery("SELECT a FROM Account a");
        List<Account> currList = q.getResultList();
        return currList;
    }

    public Account updateAccountBalance(int accNo, double newBalance) {
        Account acc = em.find(Account.class, accNo);
        if (acc == null) {
            throw new IllegalArgumentException("Account not found : Account Number ("
+ accNo + ")");
        }
        EntityTransaction et = em.getTransaction();
        et.begin();
        acc.setBalance(newBalance);
        et.commit();
        return acc;

    }

    public void deleteAccount(int accNo) {
        EntityTransaction et = em.getTransaction();
        et.begin();
        em.remove(em.getReference(Account.class, accNo));
        et.commit();
```

```
    }
}
```

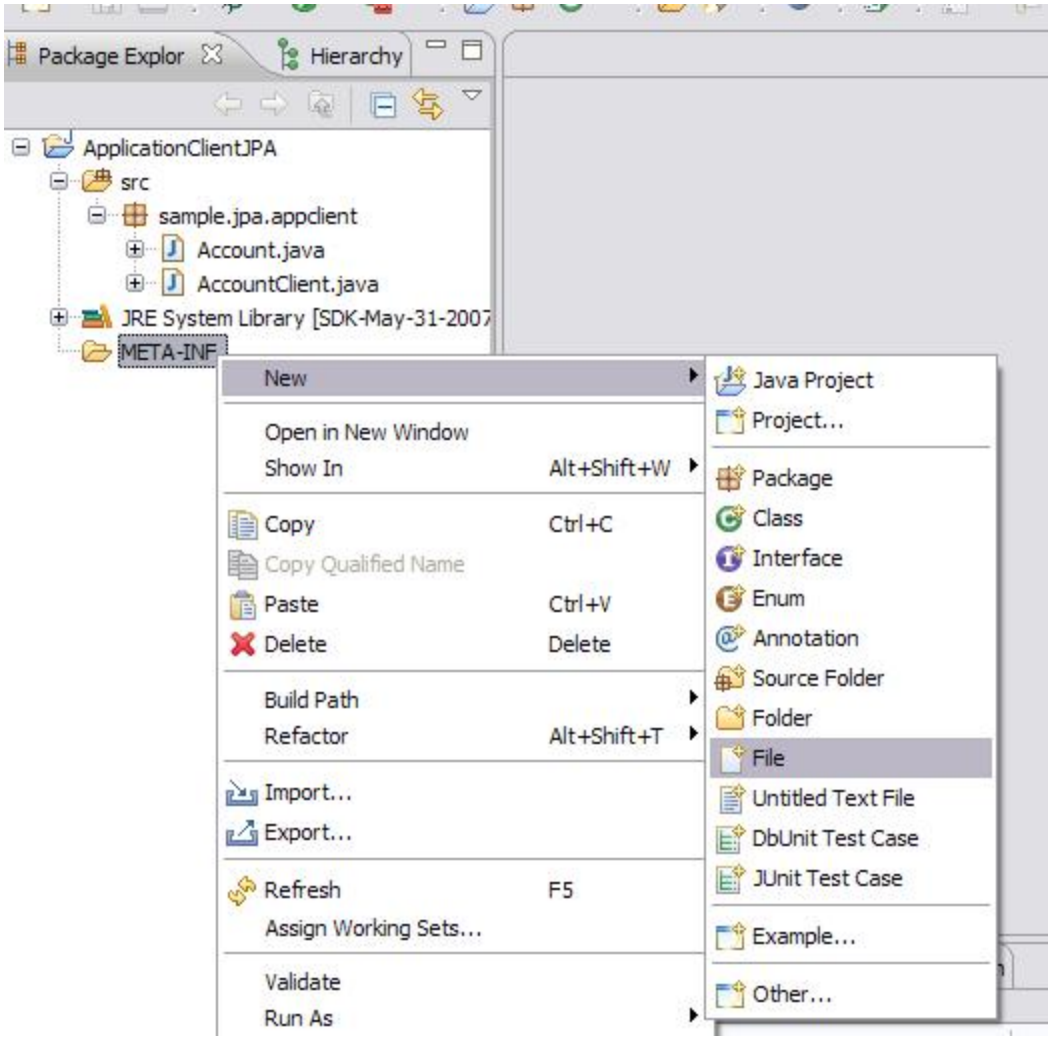
The `AccountClient` obtains `EntityManagerFactory` object first and then creates an `EntityManager` object from the factory. The `EntityManager` object thus obtained is Application Managed EntityManager object. The Persistence scope of the EntityManager is by default `Extended`. Since the application client runs in a different JVM from the server, the transaction type is `RESOURCE_LOCAL`. Hence the `AccountClient` must use `EntityTransaction` to demarcate the transactions.

Preparing Deployment Descriptors and Deployment Plans

1. Create `persistence.xml` file in the `META-INF` folder as follows.
Right click on the `META-INF` folder and click on *New* => *File*.



Provide `persistence.xml` as the file name and click on *Finish*.



Copy the below contents to the `persistence.xml`.

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
  version="1.0" >
  <persistence-unit name="JPA-App-Client">
    <description>JPA Application Client</description>
    <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
    <class>sample.jpa.appclient.Account</class>
    <properties>
      <property name="openjpa.ConnectionURL" value="jdbc:derby://localhost/AccountDB"
/>
      <property name="openjpa.ConnectionDriverName"
value="org.apache.derby.jdbc.ClientDriver" />
      <property name="ConnectionUserName" value="app" />
      <property name="openjpa.jdbc.SynchronizeMappings" value="false" />
    </properties>
  </persistence-unit>
</persistence>
```

2. Similarly, create `application-client.xml` file in the `META-INF` folder as given the previous step and copy the below contents into the file.

application-client.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application-client xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/application-client_5.xsd"
  version="5">
</application-client>
```

Since we are using JPA annotations to declare the resources, we do not require any entries in the file.

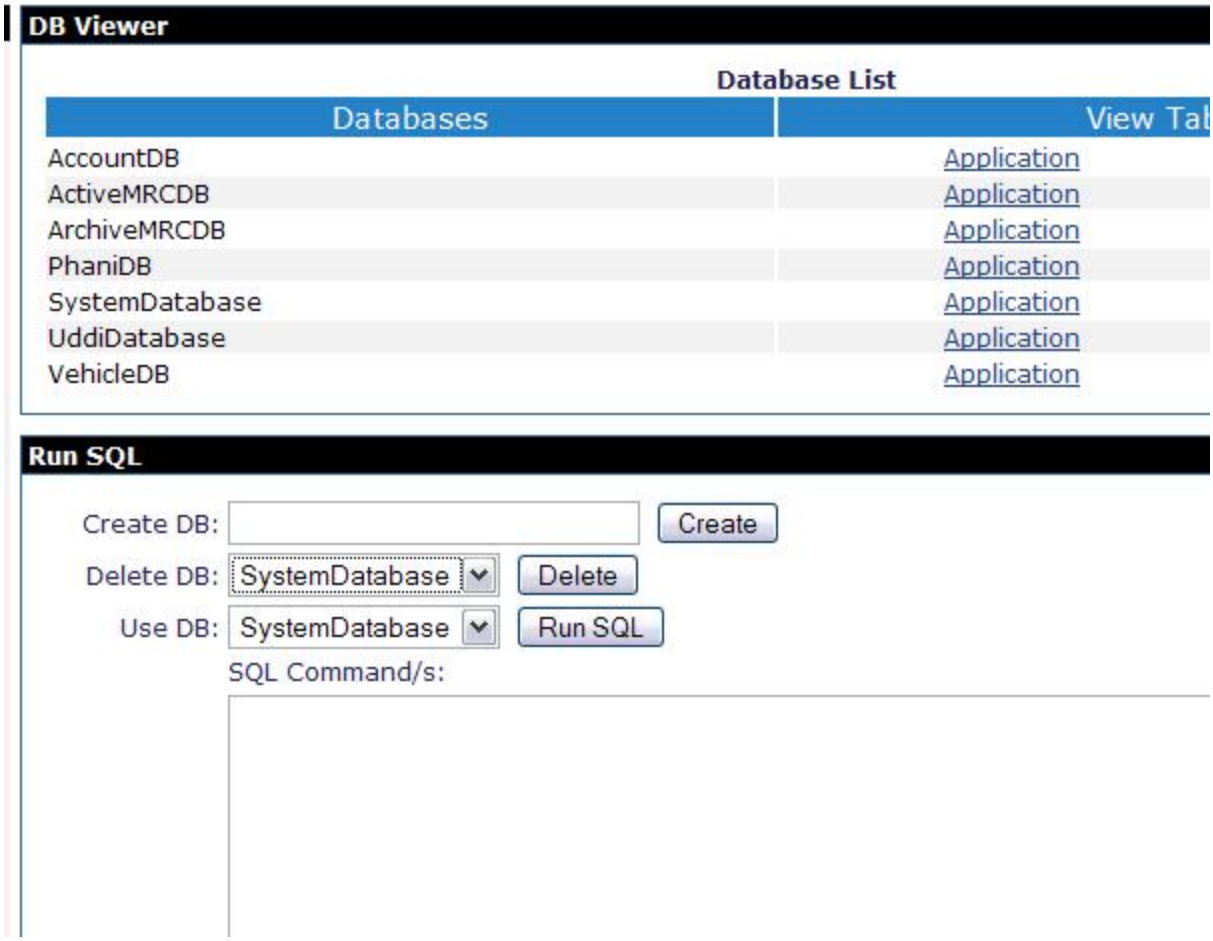
3. Similarly, create `geronimo-application-client.xml` file under `META-INF` file and copy the below contents.

geronimo-application-client.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application-client
xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-client-2.0"
  xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2"
  xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2"
  xmlns:security="http://geronimo.apache.org/xml/ns/security-2.0"
  xmlns:connector="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2">
  <sys:client-environment>
    <sys:moduleId>
      <sys:groupId>AccountJPA</sys:groupId>
      <sys:artifactId>AccountJPA-app-client</sys:artifactId>
      <sys:version>3.0</sys:version>
      <sys:type>jar</sys:type>
    </sys:moduleId>
    <sys:dependencies>
      <sys:dependency>
        <sys:groupId>org.apache.geronimo.configs</sys:groupId>
        <sys:artifactId>transaction</sys:artifactId>
        <sys:version>2.1</sys:version>
        <sys:type>car</sys:type>
      </sys:dependency>
    </sys:dependencies>
  </sys:client-environment>
  <sys:server-environment>
    <sys:moduleId>
      <sys:groupId>AccountJPA</sys:groupId>
      <sys:artifactId>AccountJPA-app-client-server</sys:artifactId>
      <sys:version>3.0</sys:version>
      <sys:type>jar</sys:type>
    </sys:moduleId>
  </sys:server-environment>
</application-client>
```

Deploying the application client

1. Export the Java Project to a jar file Start the geronimo server and open the admin console `http://localhost:8080/console`.
Click on Embedded *DB => DB Manager* on the *Console Navigation* portlet. This will open up *DB Viewer* and *Run SQL* portlets on the right hand side as follows.



Enter *AccountDB* in the *Create DB* text box and click on *Create* button. This will create a new Database *AccountDB* and will be listed in the *DB Viewer* portlet.

In the textarea *SQL Command/s:* which is on the *Run SQL* portlet, enter the below SQL command, and select *AccountDB* in the *Use DB* combo box, and click on *Run SQL* button. This creates the *Account1* table in the *AccountDB* database.

Account1 table

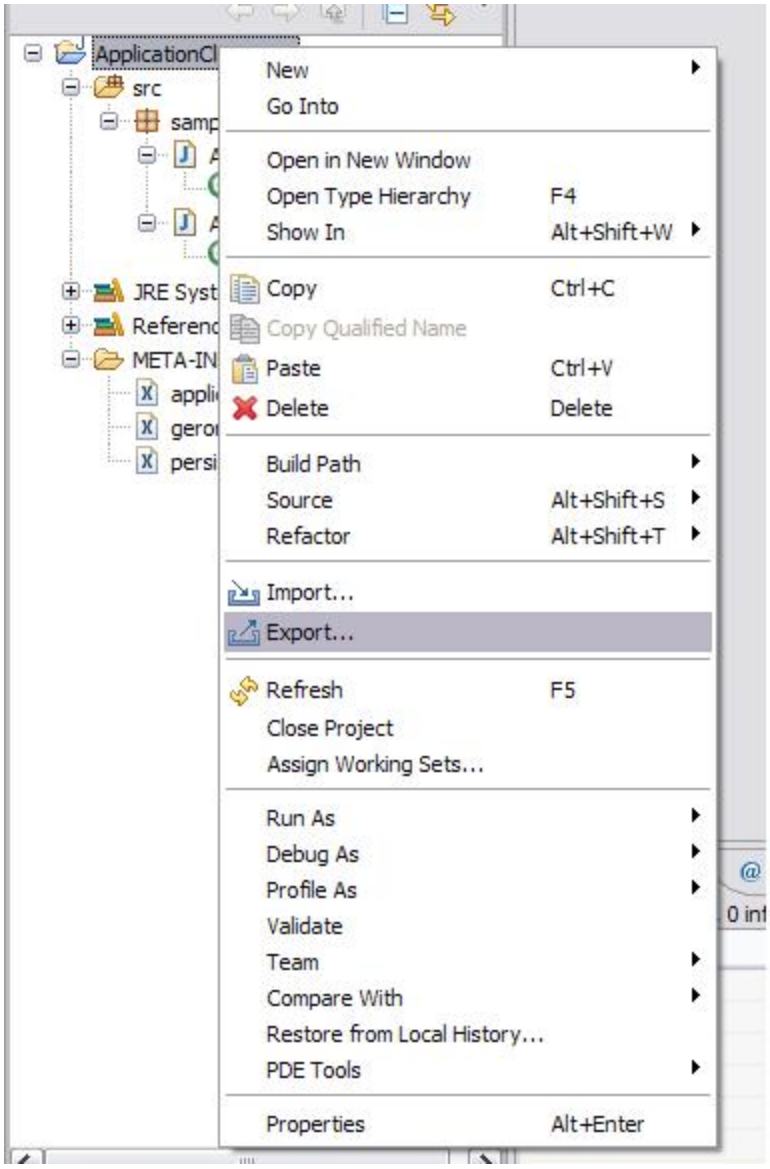
```
create table ACCOUNT1 (ACCOUNTNO integer, NAME varchar(50), ADDRESS varchar(225),
BRANCHCODE integer, BALANCE decimal(15,2));
```

Insert some sample rows in the table using the below SQL statements by following the same outlined procedure above.

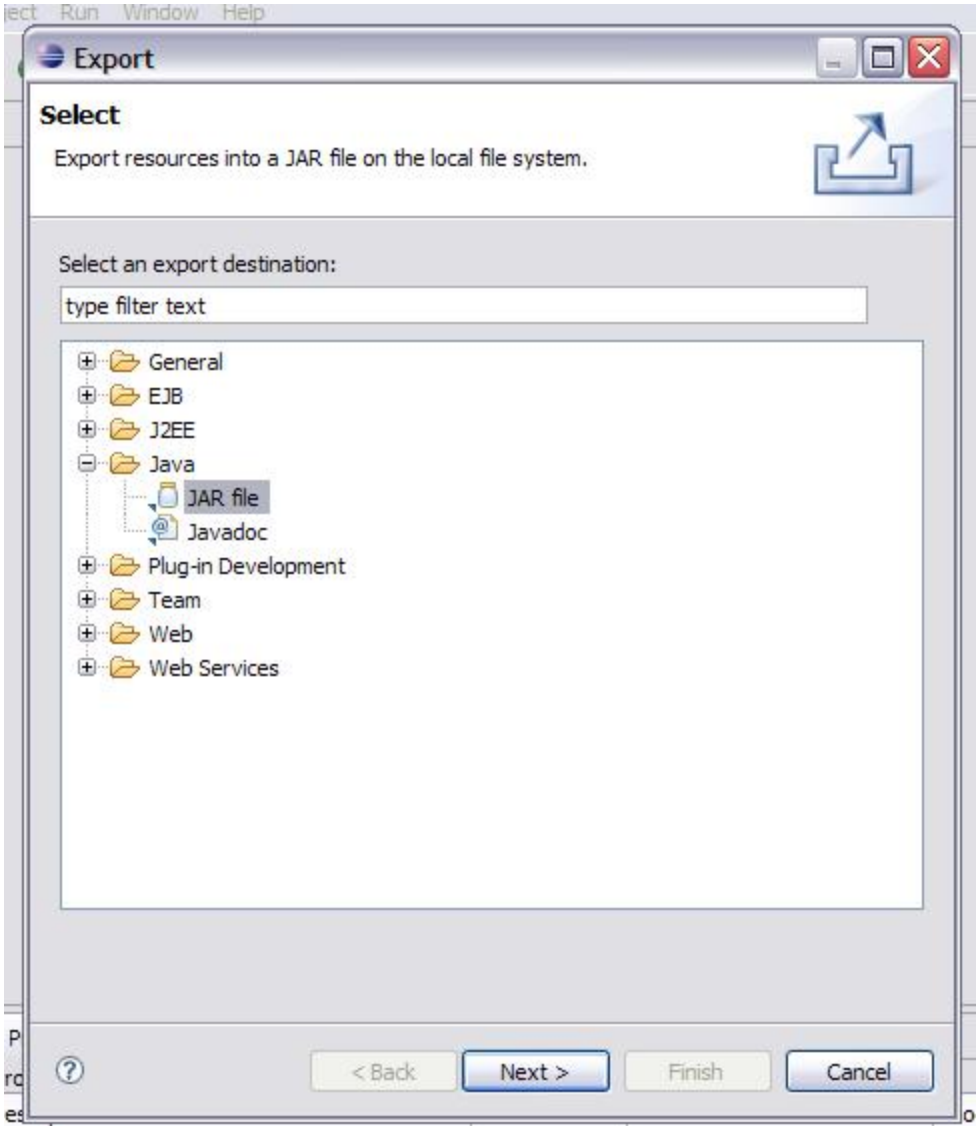
Account1 table

```
insert into Account1 values (1111, 'Joe', 'NewYork', 10, 30000.0);
insert into Account1 values (2222, 'John', 'NewJersy', 11, 31000.0);
insert into Account1 values (3333, 'Jane', 'Raleigh', 13, 32000.0);
```

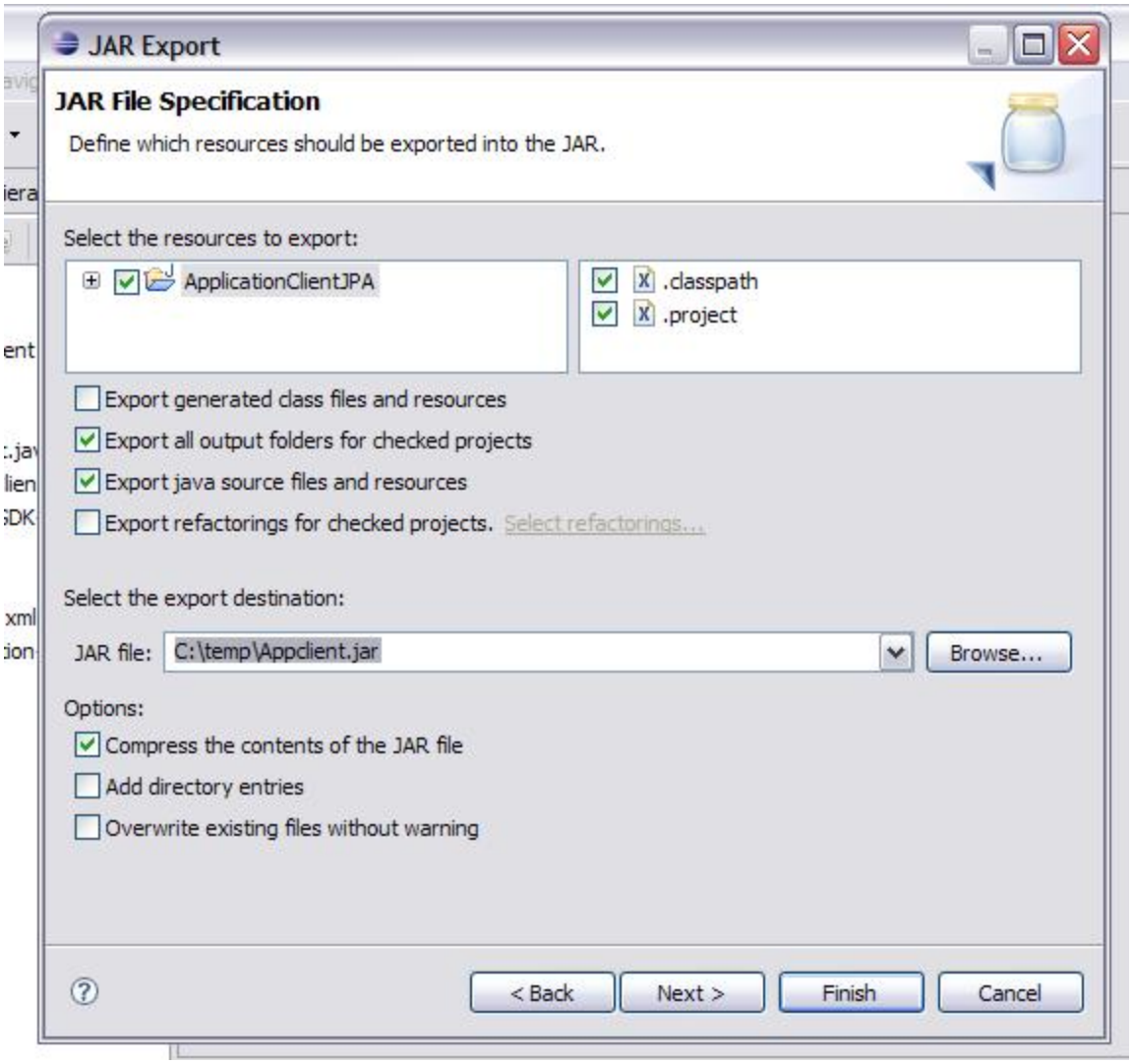
2. Export the Java project by name `ApplicationClientJPA.jar` as follows.
Right click on the java project and export it as jar file.



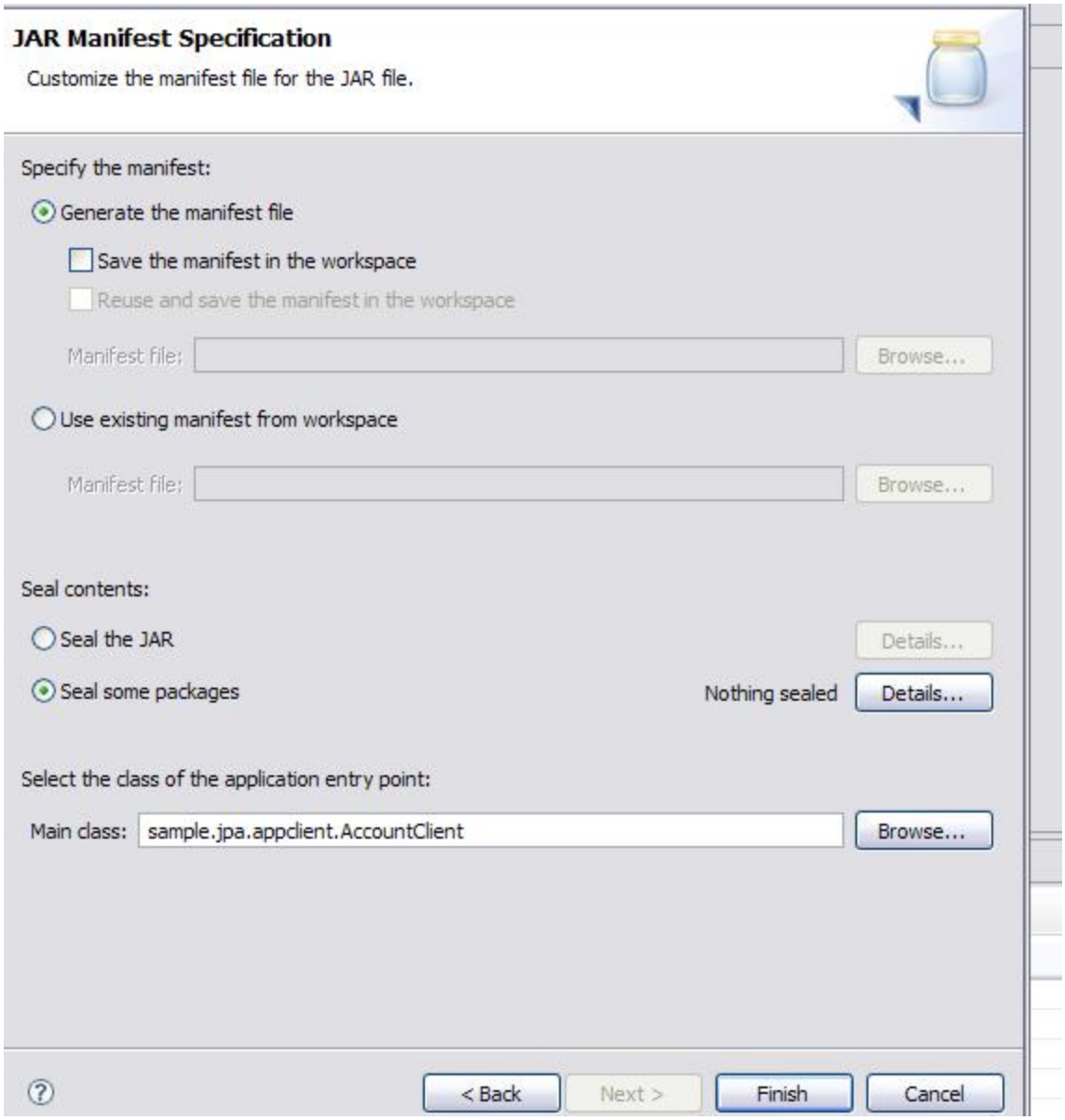
Select *Java => jar*



Click on *Next* and provide the jar file name and destination folder as below and click on *Next*.



In the next screen click *Next*. In the *Jar Manifest Specification* wizard, select `sample.jpa.appclient.AccountClient` using *Browse* button. This is the class that has `public static void main(String[] args)` method which is the application entry point.



Click *Finish*.

3. Deploy the application client as follows.

Change the directory to `<geronimo_home>\bin`.
Submit the below command

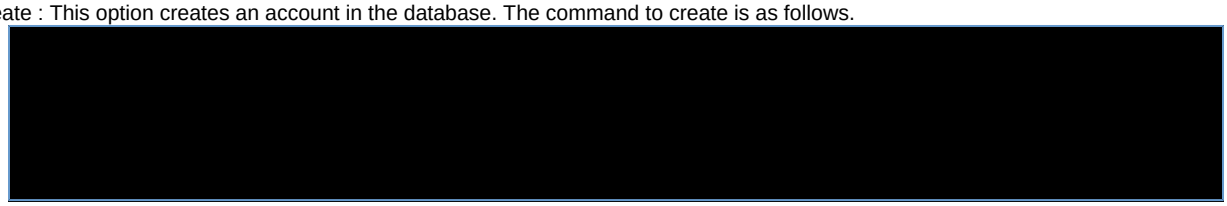
Running the application client

1. Run the application client as follows.

The application performs the following operations. Use the script `client` (.bat or .sh) to run the application client.

- List : This option lists the accounts curently in the database. The command to list is as follows.

- Create : This option creates an account in the database. The command to create is as follows.



- Update : This option updates an account with a new balance in the database.
  The command to update is as follows.

- Delete : This option deletes an account in the database. The command to delete is as follows.