BankAccount.java

```
package org.bank;

import javax.persistence.DiscriminatorColumn;

import javax.persistence.DiscriminatorType;

import javax.persistence.DiscriminatorValue;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Inheritance;

import javax.persistence.InheritanceType;

@Entity

@Inheritance(strategy = InheritanceType.SINGLE_TABLE)

@DiscriminatorColumn(name = "account", discriminatorType = DiscriminatorType.STRING)

@DiscriminatorValue("bank")

public class BankAccount {

@Id

@GeneratedValue

private long accountNumber;

private String accountHolder;

private String address;

private long phoneNumber;

private String emailId;

protected double balance;

public BankAccount() {

super();
```

```
// TODO Auto-generated constructor stub
}
public BankAccount(long accountNumber, String accountHolder, String address,
long phoneNumber, String emailId,
double balance) {
super();
this.accountNumber = accountNumber;
this.accountHolder = accountHolder;
this.address = address;
this.phoneNumber = phoneNumber;
this.emailId = emailId;
this.balance = balance;
}
public long getAccountNumber() {
return accountNumber;
}
public void setAccountNumber(long accountNumber) {
this.accountNumber = accountNumber;
}
public String getAccountHolder() {
return accountHolder;
}
public void setAccountHolder(String accountHolder) {

this.accountHolder = accountHolder;
```

```
}
public String getAddress() {
return address;
}
public void setAddress(String address) {
this.address = address;
}
public long getPhoneNumber() {
return phoneNumber;
}
public void setPhoneNumber(long phoneNumber) {
this.phoneNumber = phoneNumber;
}
public String getEmailId() {
return emailId;
}
public void setEmailId(String emailId) {
this.emailId = emailId;
}
public double getBalance() {
return balance;
}
public void setBalance(double balance) {
this.balance = balance;
}
```

```
public Double withdraw(double amount) {
return this.balance - amount;
}
public Double deposit(double amount) {
return this.balance + amount;
}
}
```

Savings.java

```
package org.bank;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
@Entity
@DiscriminatorValue("savings")
public class Savings extends BankAccount {
private static double maximumAmountTransfer = 100000;

private static int maximumNumberOfTransaction = 5;
private double amountTransferred;
private int numberOfTransaction;
public Savings() {
super();
// TODO Auto-generated constructor stub
}
public Savings(long accountNumber, String accountHolder, String address, long
phoneNumber, String emailId,
```

```
double balance, double amountTransferred, int

numberOfTransaction) {

super(accountNumber, accountHolder, address, phoneNumber, emailId,

balance);

this.amountTransferred = amountTransferred;
this.numberOfTransaction = numberOfTransaction;
// TODO Auto-generated constructor stub
}
public static double getMaximumAmountTransfer() {
return maximumAmountTransfer;
}
public static void setMaximumAmountTransfer(double maximumAmountTransfer) {
Savings.maximumAmountTransfer = maximumAmountTransfer;
}
public static int getMaximumNumberOfTransaction() {
return maximumNumberOfTransaction;
}
public static void setMaximumNumberOfTransaction(int
maximumNumberOfTransaction) {
```

```
Savings.maximumNumberOfTransaction = maximumNumberOfTransaction;
}
public double getAmountTransferred() {
return amountTransferred;
}
public void setAmountTransferred(double amountTransferred) {
this.amountTransferred = amountTransferred;
}
public int getNumberOfTransaction() {
return numberOfTransaction;
}
public void setNumberOfTransaction(int numberOfTransaction) {
this.numberOfTransaction = numberOfTransaction;
}
@Override
public Double withdraw(double amount) {

if (maximumAmountTransfer <= 100000  && maximumNumberOfTransaction <= 5)

{

return super.withdraw(amount);
} else {
System.out.println("process cannot been done");
}
```

```
return amount;

}

@Override

public Double deposit(double amount) {

if (maximumAmountTransfer <= 100000  && maximumNumberOfTransaction <= 5)



{



return super.deposit(amount);

} else {

System.out.println("process cannot been done");

}

return amount;

}

}
```

Current.java

```
package org.bank;

import javax.persistence.DiscriminatorValue;

import javax.persistence.Entity;

@Entity

@DiscriminatorValue("current")

public class Current extends BankAccount {

private static double minimumAmountTransfer = 500000;

private static int minimumNoOfTransaction = 7;

private double amountTransferred;
```

```
private int NoOfTransactionHeld;
public Current() {
super();
// TODO Auto-generated constructor stub
}
public Current(long accountNumber, String accountHolder, String address, long
phoneNumber, String emailId,

double balance, double minimumAmountTransfer, int

minimumNoOfTransaction, double amountTransferred,
int NoOfTransactionHeld) {
super(accountNumber, accountHolder, address, phoneNumber, emailId,

balance);

// TODO Auto-generated constructor stub
}
public static double getMinimumAmountTransfer() {
return minimumAmountTransfer;
}

public static void setMinimumAmountTransfer(double minimumAmountTransfer) {
Current.minimumAmountTransfer = minimumAmountTransfer;
}
```

```
public static int getMinimumNoOfTransaction() {
return minimumNoOfTransaction;
}
public static void setMinimumNoOfTransaction(int minimumNoOfTransaction) {
Current.minimumNoOfTransaction = minimumNoOfTransaction;
}
public double getAmountTransferred() {
return amountTransferred;
}
public void setAmountTransferred(double amountTransferred) {
this.amountTransferred = amountTransferred;
}
public int getNoOfTransactionHeld() {
return NoOfTransactionHeld;
}
public void setNoOfTransactionHeld(int noOfTransactionHeld) {
NoOfTransactionHeld = noOfTransactionHeld;
}
@Override
public Double withdraw(double amount) {
if (minimumAmountTransfer <= 500000  && minimumNoOfTransaction <= 7) {
return super.withdraw(amount);
} else {
System.out.println("process cannot been done");
}
```

```
return amount;
}
@Override
public Double deposit(double amount) {
if (minimumAmountTransfer <= 500000  && minimumNoOfTransaction <= 7) {
return super.deposit(amount);
} else {
System.out.println("process cannot been done");
}
return amount;
}
}
package org.bank;
import java.io.BufferedReader;
```

Solution.java

```
import java.io.IOException;
import java.io.InputStreamReader;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class Solution {
public static void main(String[] args) throws NumberFormatException,
IOException {

SessionFactory sf = new
```

```
Configuration().configure().buildSessionFactory();
Session session = sf.openSession();
session.beginTransaction();
BufferedReader bf = new BufferedReader(new

InputStreamReader(System.in));

System.out.println("enter account number");
long accountNumber = Long.valueOf(bf.readLine());
System.out.println("enter account holder");
String accountHolder = bf.readLine();
System.out.println("enter address");
String address = bf.readLine();
System.out.println("enter phone number");
long phoneNumber = Long.valueOf(bf.readLine());
System.out.println("enter emailid");
String emailId = bf.readLine();
System.out.println("enter balance");
double balance = Double.valueOf(bf.readLine());
System.out.println("enter amount");
double amount = Double.valueOf(bf.readLine());
BankAccount bankaccount = new BankAccount(accountNumber, accountHolder,

address, phoneNumber, emailId, balance);
```

```
System.out.println("the deposited value is:" +

bankaccount.deposit(amount));

System.out.println("the withdrawed value is:" +

bankaccount.withdraw(amount));

System.out.println("enter amount transferd");
double amountTransferred = Double.valueOf(bf.readLine());
System.out.println("enter number of transaction");
int numberOfTransaction = Integer.valueOf(bf.readLine());
Savings savings = new Savings(accountNumber, accountHolder, address,

phoneNumber, emailId, balance,

amountTransferred, numberOfTransaction);

System.out.println("the deposited value is:" + savings.deposit(amount));
System.out.println("the withdrawed value is:" +

savings.withdraw(amount));

System.out.println("enter number of transaction held");
```

```
int noOfTransactionHeld = Integer.valueOf(bf.readLine());
Current current = new Current(accountNumber, accountHolder, address,

phoneNumber, emailId, balance,

amountTransferred, noOfTransactionHeld, amountTransferred,

noOfTransactionHeld);

System.out.println("the deposited value is:" + current.deposit(amount));

System.out.println("the withdrawed value is:" +

current.withdraw(amount));

System.out.println(bankaccount.getAccountNumber());
System.out.println(bankaccount.getAddress());
System.out.println(bankaccount.getPhoneNumber());
System.out.println(bankaccount.getEmailId());
System.out.println(bankaccount.getBalance());
System.out.println(savings.getAmountTransferred());
System.out.println(savings.getNumberOfTransaction());
System.out.println(current.getAmountTransferred());
System.out.println(current.getNoOfTransactionHeld());
session.save(bankaccount);
```

```
session.save(savings);

session.save(current);

session.getTransaction().commit();

session.close();

}

}
```

Hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>

<!-- ~ Hibernate, Relational Persistence for Idiomatic Java ~ ~ License:

GNU Lesser General Public License (LGPL), version 2.1 or later. ~ See the

lgpl.txt file in the root directory or <http://www.gnu.org/licenses/lgpl-

2.1.html>. -->

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- Database connection settings -->

<property


name="connection.driver_class">com.mysql.jdbc.Driver</property>


<property


name="connection.url">jdbc:mysql://localhost:3306/sample</property>
```

```
<property name="connection.username">root</property>

<property name="connection.password"></property>

<!-- JDBC connection pool (use the built-in) -->

<property name="connection.pool_size">10</property>

<!-- SQL dialect -->

<property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>

<!-- Disable the second-level cache -->

<property


name="cache.provider_class">org.hibernate.cache.internal.NoCacheProvider</property>


<!-- Echo all executed SQL to stdout -->

<property name="show_sql">true</property>


<!-- Drop and re-create the database schema on startup -->

<property name="hbm2ddl.auto">create</property>

<!-- Names the annotated entity class -->

<mapping class="org.bank.BankAccount" />

<mapping class="org.bank.Savings" />

<mapping class="org.bank.Current" />

</session-factory>

</hibernate-configuration>
```