Step 1
Get an informal list of the responsibilities of your objects.

The following responsibilities are mentioned in the problem statement:

- Deposit funds.
- Withdraw funds.
- Add interest.

There is a hidden responsibility as well. We need to be able to find out how much money is in the account.
- Get balance.

Step 2
Specify the public interface.

We need to supply parameter variables and determine which methods are accessors and mutators.

To deposit or withdraw money, one needs to know the amount of the deposit or withdrawal:

- void deposit(double amount)
- void withdraw(double amount)

To add interest, one needs to know the interest rate that is to be applied:
- void addInterest(double rate)

Clearly, all these methods are mutators because they change the balance.

Finally, we have:
- double getBalance()

This method is an accessor because inquiring about the balance does not change it.

Now we move on to constructors. The constructor with no arguments makes an account with a zero balance. It can also be useful to supply a constructor with an initial balance.

Here is the complete public interface:

- Constructors
  - public BankAccount()
  - public BankAccount(double initialBalance)

- Mutators
  - public void deposit(double amount)
  - public void withdraw(double amount)
  - public void addInterest(double rate)

- Accessors
  - public double getBalance()

Step 3
Document the public interface.

```java
/**
   * A bank account has a balance that can be changed by deposits and withdrawals.
   */
public class BankAccount
```

```java
}
```
/**
 * Constructs a bank account with a zero balance.
 * 
 * @return
 */
public BankAccount()

/**
 * Constructs a bank account with a given balance.
 * @param initialBalance the initial balance
 */
public BankAccount(double initialBalance)

/**
 * Deposits money into this account.
 * @param amount the amount to deposit
 */
public void deposit(double amount)

/**
 * Makes a withdrawal from this account, or charges a penalty if sufficient funds are not available.
 * @param amount the amount of the withdrawal
 */
public void withdraw(double amount)

/**
 * Adds interest to this account.
 * @param rate the interest rate in percent
 */
public void addInterest(double rate)

/**
 * Gets the current balance of this account.
 * @return the current balance
 */
public double getBalance()

Step 4 Determine instance variables.

Clearly we need to store the bank balance.

```java
public class BankAccount {
    private double balance;
    ...
}
```

Do we need to store the interest rate? No—it varies every month, and is supplied as an argument to `addInterest`. What about the withdrawal penalty? The problem description states that it is a fixed $10, so we need not store it. If the penalty could vary over time, as is the case with most real bank accounts, we would need to store it somewhere (perhaps in a `Bank` object), but it is not our job to model every aspect of the real world.

Step 5 Implement constructors and methods.

Let's start with a simple one:

```java
public double getBalance() {
    return balance;
}
```
The deposit method is a bit more interesting:

```java
public void deposit(double amount)
{
    balance = balance + amount;
}
```

The withdraw method needs to charge a penalty if sufficient funds are not available:

```java
public void withdraw(double amount)
{
    final double PENALTY = 10;
    if (amount > balance)
    {
        balance = balance - PENALTY;
    }
    else
    {
        balance = balance - amount;
    }
}
```

Finally, here is the addInterest method. We compute the interest and then add it to the balance:

```java
public void addInterest(double rate)
{
    double amount = balance * rate / 100;
    balance = balance + amount;
}
```

The constructors are once again quite simple:

```java
public BankAccount()
{
    balance = 0;
}

public BankAccount(double initialBalance)
{
    balance = initialBalance;
}
```

This finishes the implementation (see worked_example_1/BankAccount.java in your book code).

**Step 6** Test your class.

Here is a simple test program that exercises all methods:

```java
public class BankAccountTester
{
    public static void main(String[] args)
    {
        BankAccount harrysAccount = new BankAccount(1000);
        harrysAccount.deposit(500); // Balance is now $1500
        harrysAccount.withdraw(2000); // Balance is now $1490
        harrysAccount.addInterest(1); // Balance is now $1490 + 14.90
        System.out.printf("%.2f\n", harrysAccount.getBalance());
        System.out.println("Expected: 1504.90");
    }
}
```

**Program Run**

```
1504.90
Expected: 1504.90
```