CS 102
Object Oriented Programming

# Access Modifiers

Reyyan Yeniterzi
reyyan.yeniterzi@ozyegin.edu.tr

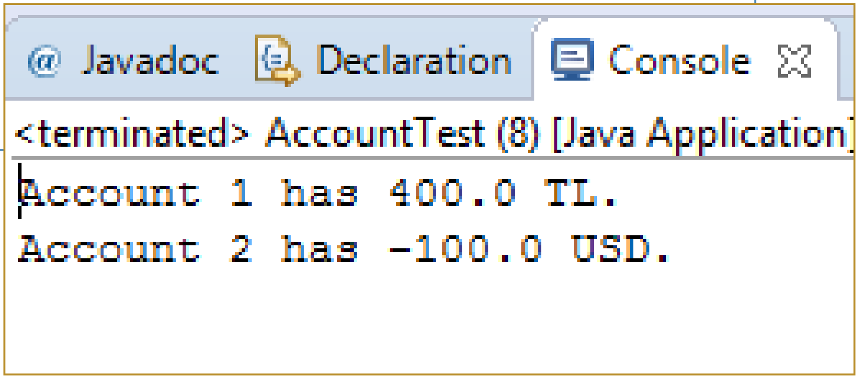
October 3, 2016

## Bank Account – version 9

```
public class AccountTest {
    public static void main(String[] args) {

        Account account1 = new Account(1, 100, "TL");
        Account account2 = new Account(2, 200, "USD");

        account1.deposit(300);
        account2.deposit(-300);

        account1.report();
        account2.report();
    }
}
```

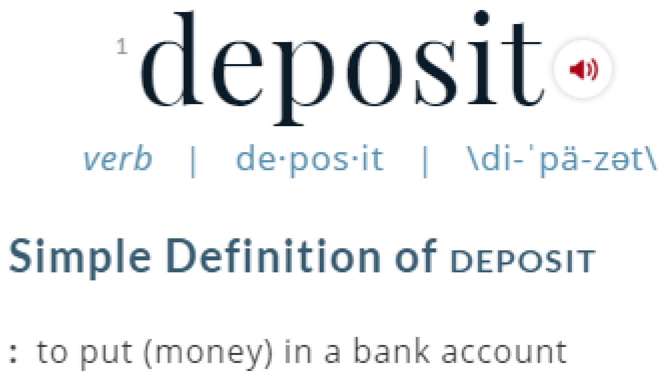
# Bank Account – version 9

```java
public class AccountTest {
    public static void main(String[] args) {

        Account account1 = new Account(1, 100, "TL");
        Account account2 = new Account(2, 200, "USD");

        account1.deposit(300);
        account2.deposit(-300);

        account1.report();
        account2.report();
    }
}
```



```
<terminated> AccountTest (8) [Java Application]
Account 1 has 400.0 TL.
Account 2 has -100.0 USD.
```

# Definition of deposit



¹deposit

*verb* | de·pos·it | \di-ˈpä-zət\

**Simple Definition of DEPOSIT**

: to put (money) in a bank account

- We should not allow depositing negative amount of money.
- How?

Source: http://www.merriam-webster.com/dictionary/deposit

## deposit function

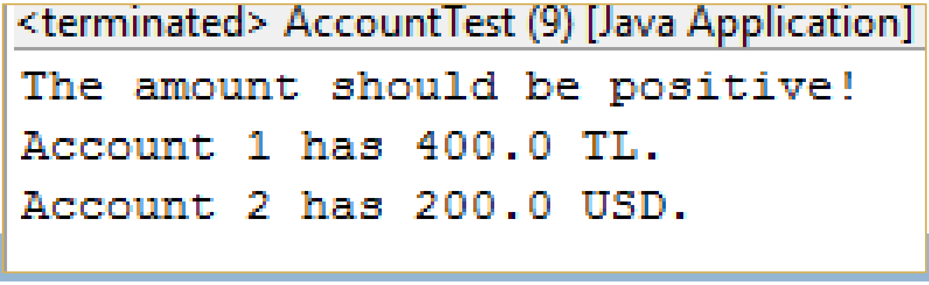
```
public void deposit(double d) {
    if (d > 0)
        balance = balance + d;
    else
        System.out.println("The amount should be positive!");
}
```

# deposit function

```
public void deposit(double d) {
    if (d > 0)
        balance = balance + d;
    else
        System.out.println("The amount should be positive!");
}
```

```
public class AccountTest {
    public static void main(String[] args) {

        Account account1 = new Account(1, 100, "TL");
        Account account2 = new Account(2, 200, "USD");

        account1.deposit(300);
        account2.deposit(-300);

        account1.report();
        account2.report();
    }
}
```

# deposit function



```
<terminated> AccountTest (9) [Java Application]
The amount should be positive!
Account 1 has 400.0 TL.
Account 2 has 200.0 USD.
```

```java
public void deposit(double d) {
    if (d > 0)
        balance = balance + d;
    else
        System.out.println("The amount should be positive!");
}
```

```java
public class AccountTest {
    public static void main(String[] args) {

        Account account1 = new Account(1, 100, "TL");
        Account account2 = new Account(2, 200, "USD");

        account1.deposit(300);
        account2.deposit(-300);

        account1.report();
        account2.report();
    }
}
```

## Bank Account

- Can you think of any other controls that we should have?

## Bank Account

- Can you think of any other controls that we should have?
- A bank account should get **a number during initialization.**
- A bank account should not have **negative initial balance.**

## Constructors

- Assume that we don’t have the **interest rate**
- We have the following constructors:

```
public Account() {

}
public Account(int n, double b, String c) {
    number = n;
    balance = b;
    currency = c;
}
public Account(int n, String c) {
    number = n;
    balance = 0;
    currency = c;
}
public Account(int n) {
    number = n;
    balance = 0;
    currency = "TL";
}
```

## Constructors

- A bank account should get **a number during initialization.**

```
public Account() {

}
public Account(int n, double b, String c) {
    number = n;
    balance = b;
    currency = c;
}
public Account(int n, String c) {
    number = n;
    balance = 0;
    currency = c;
}
public Account(int n) {
    number = n;
    balance = 0;
    currency = "TL";
}
```

## Constructors

- A bank account should get **a number during initialization.**
- Remove the following constructor.

```
public Account()  {

}
```

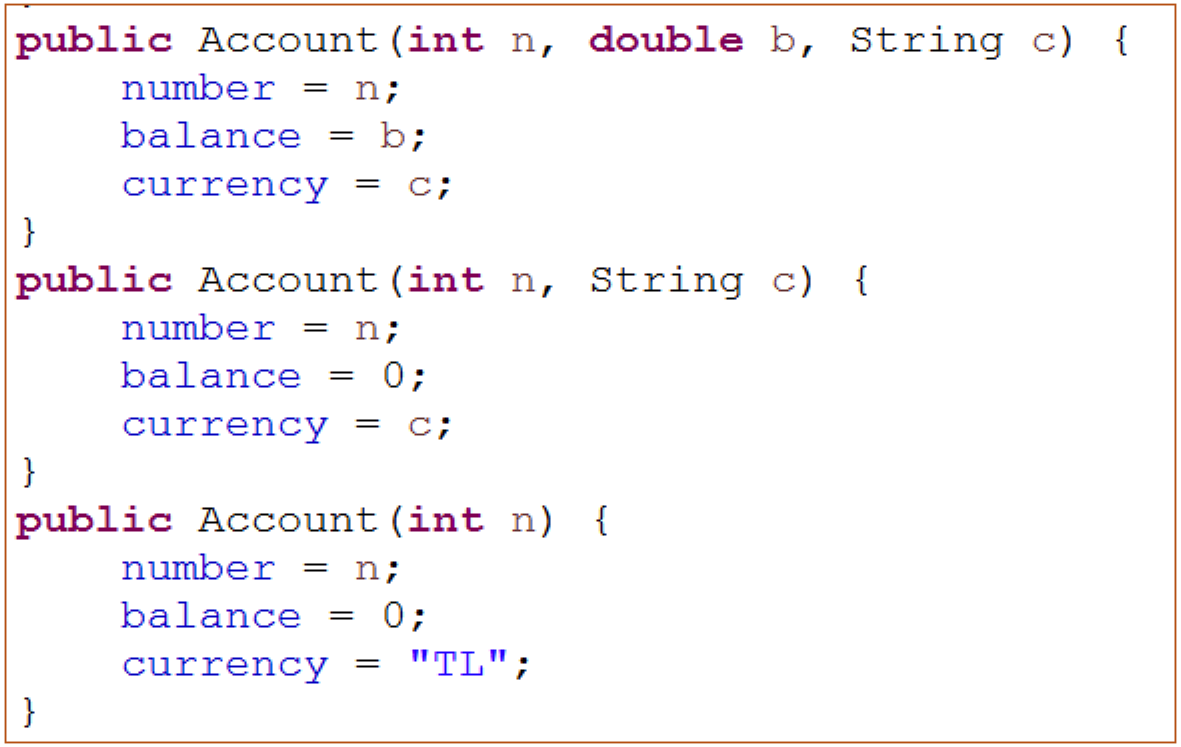
## Constructors

- A bank account should get **a number during initialization.**

```
public Account(int n, double b, String c) {
     number = n;
     balance = b;
     currency = c;
}
public Account(int n, String c) {
     number = n;
     balance = 0;
     currency = c;
}
public Account(int n) {
     number = n;
     balance = 0;
     currency = "TL";
}
```

## Constructors

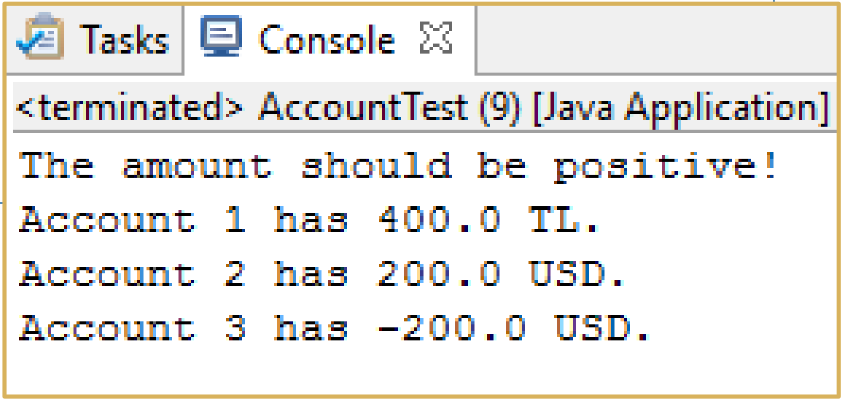
- A bank account should not have **negative initial balance.**



```
public Account(int n, double b, String c) {
     number = n;
     balance = b;
     currency = c;
}
public Account(int n, String c) {
     number = n;
     balance = 0;
     currency = c;
}
public Account(int n) {
     number = n;
     balance = 0;
     currency = "TL";
}
```

# Negative Initial Balance

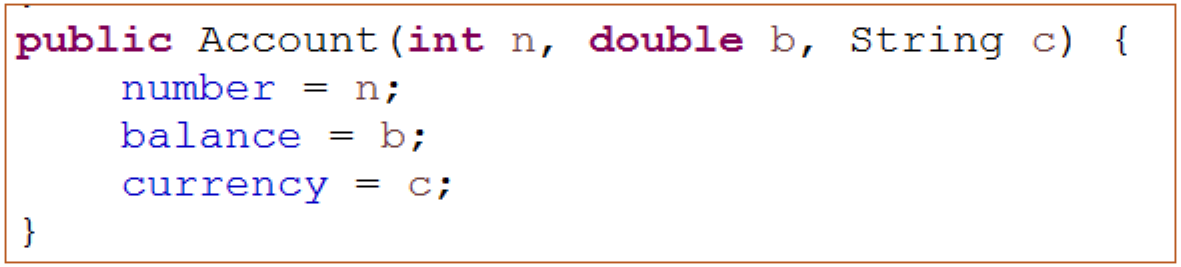
```java
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");
    Account account3 = new Account(3, -200, "USD");

    account1.deposit(300);
    account2.deposit(-300);

    account1.report();
    account2.report();
    account3.report();
}
```



```
<terminated> AccountTest (9) [Java Application]
The amount should be positive!
Account 1 has 400.0 TL.
Account 2 has 200.0 USD.
Account 3 has -200.0 USD.
```

# Constructors

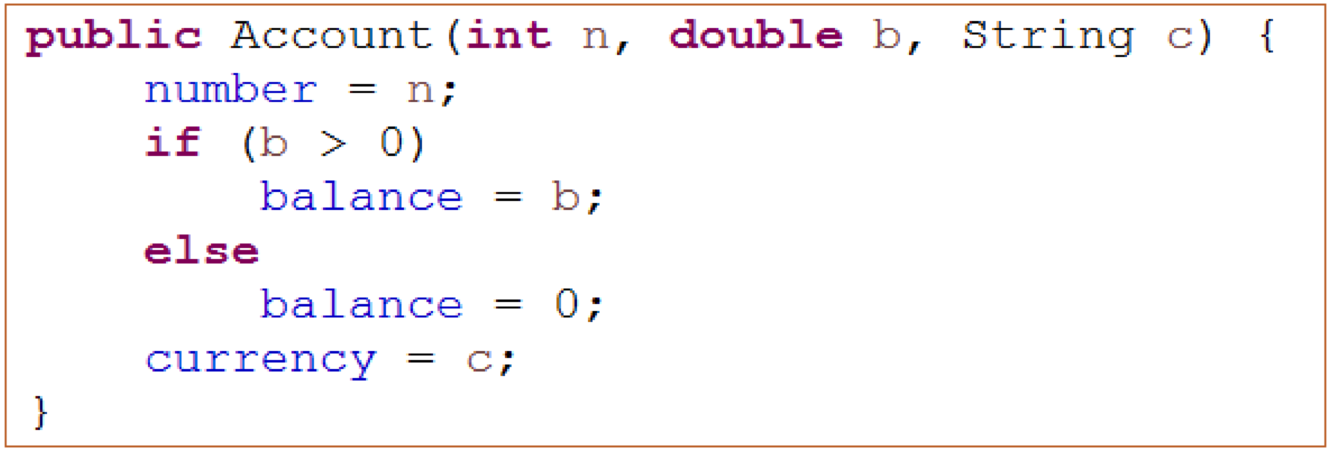
- A bank account should not have **negative initial balance.**
- We should have check the initial balance.



```
public Account(int n, double b, String c) {
    number = n;
    balance = b;
    currency = c;
}
```

## Constructors

- A bank account should not have **negative initial balance.**
- We should have check the initial balance.
- If it is negative, the balance should be 0.



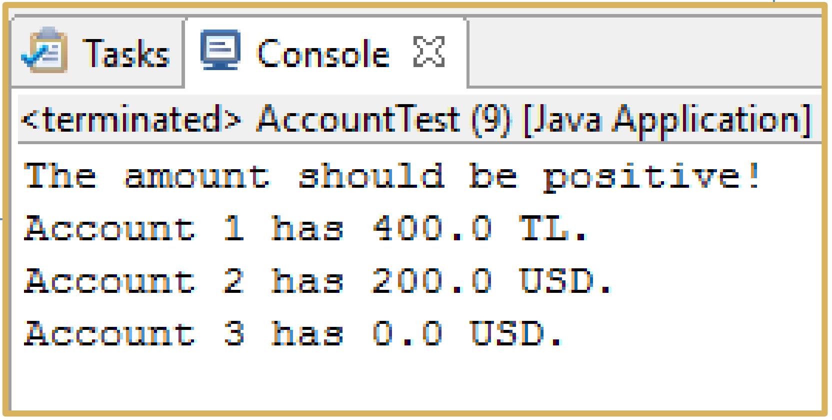
```
public Account(int n, double b, String c) {
    number = n;
    if (b > 0)
        balance = b;
    else
        balance = 0;
    currency = c;
}
```

# What is the output?

```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");
    Account account3 = new Account(3, -200, "USD");

    account1.deposit(300);
    account2.deposit(-300);

    account1.report();
    account2.report();
    account3.report();
}
```

## What is the output?

```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");
    Account account3 = new Account(3, -200, "USD");

    account1.deposit(300);
    account2.deposit(-300);

    account1.report();
    account2.report();
    account3.report();
}
```


Tasks Console

<terminated> AccountTest (9) [Java Application]

```
The amount should be positive!
Account 1 has 400.0 TL.
Account 2 has 200.0 USD.
Account 3 has 0.0 USD.
```

## So, are we done?

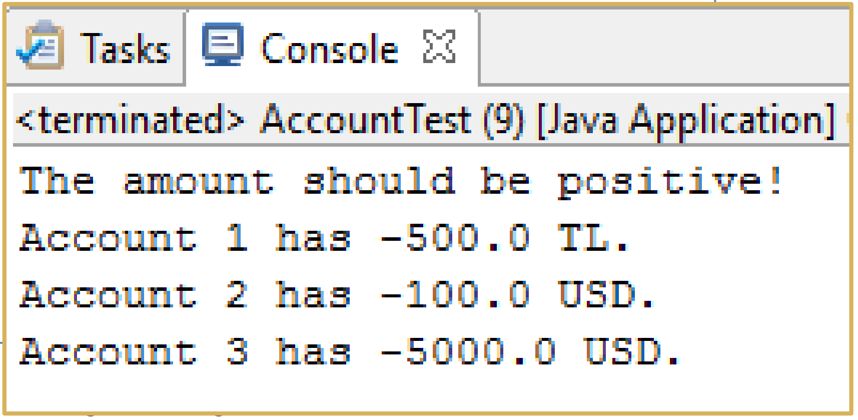
- With changing the constructor and the deposit function, are we sure that balance will not be a negative amount?

# What is the output?

```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");
    Account account3 = new Account(3, -200, "USD");

    account1.deposit(300);
    account2.deposit(-300);

    account1.balance = -500;
    account2.balance = -100;
    account3.balance = -5000;

    account1.report();
    account2.report();
    account3.report();
}
```

# What is the output?

```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");
    Account account3 = new Account(3, -200, "USD");

    account1.deposit(300);
    account2.deposit(-300);

    account1.balance = -500;
    account2.balance = -100;
    account3.balance = -5000;

    account1.report();
    account2.report();
    account3.report();
}
```



```
<terminated> AccountTest (9) [Java Application]
The amount should be positive!
Account 1 has -500.0 TL.
Account 2 has -100.0 USD.
Account 3 has -5000.0 USD.
```

# Class instances

- The class instances need to be protected.
- We need to keep the control of how these instances are accessed.
- How?

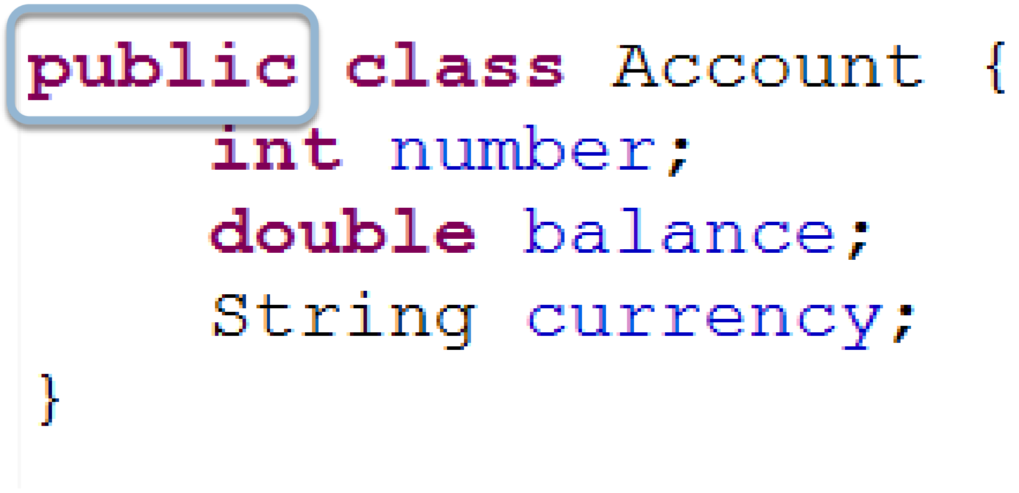
# Class instances

- The class instances need to be protected.
- We need to keep the control of how these instances are accessed.
- How?
- Through using access modifiers.

## Access Modifier

- They are used to set access levels for classes, variables, and other entries.

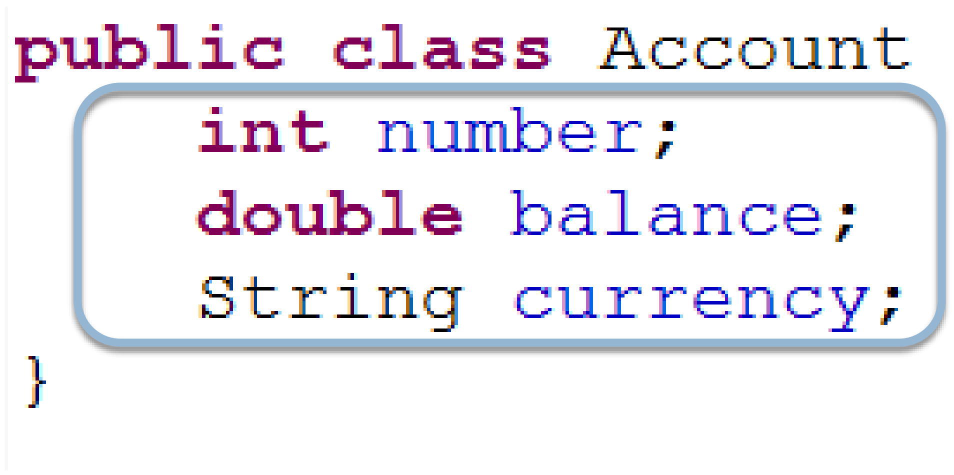
# Access Specification



```
public class Account {
        int number;
        double balance;
        String currency;
}
```

- Access modifier
  - For the top level classes it can be either
    - **public** or : visible to the earth
    - **default** (no keyword) : visible only within the same package

# Access Specification



```
public class Account {
    int number;
    double balance;
    String currency;
}
```

- These variables don’t have any particular access modifier, therefore they are visible …?

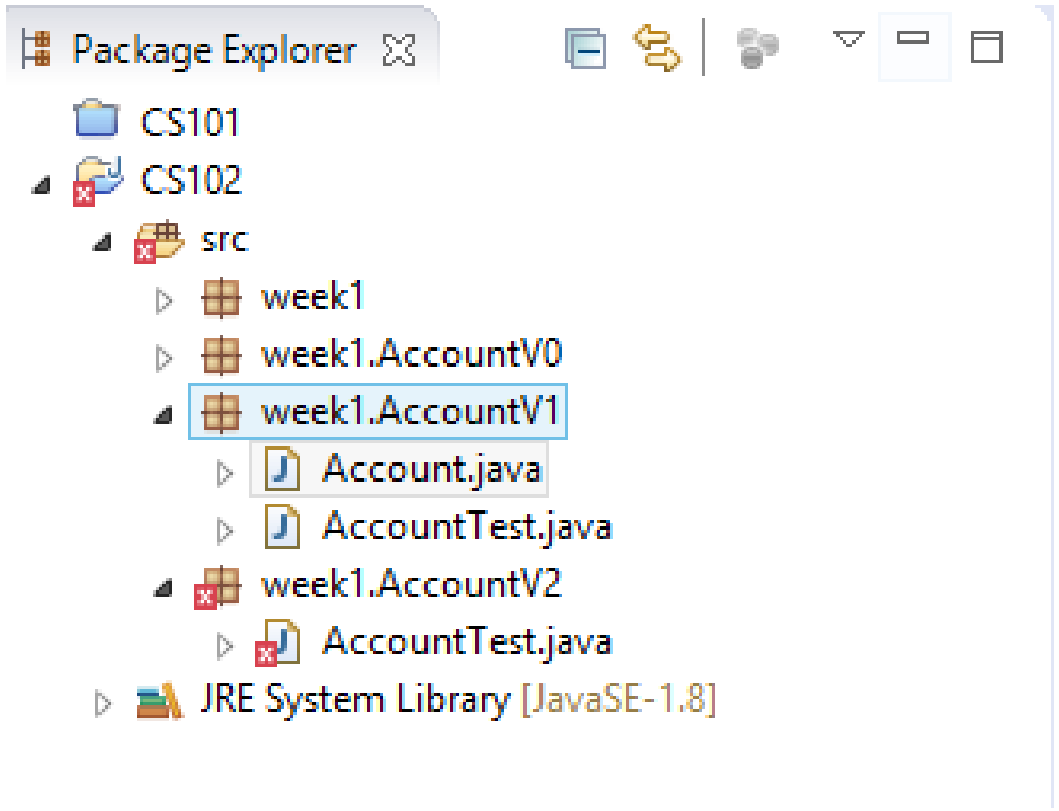
## Access Specification

```
public class Account {
    int number;
    double balance;
    String currency;
}
```

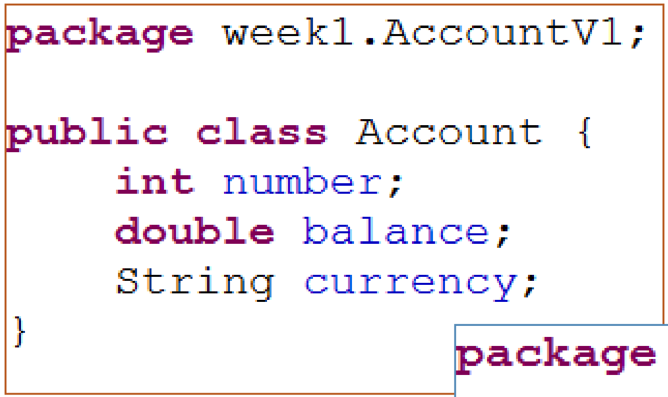
- These variables don’t have any particular access modifier, therefore they are visible and accessible from only within the same package (package-private).

## Access Specification

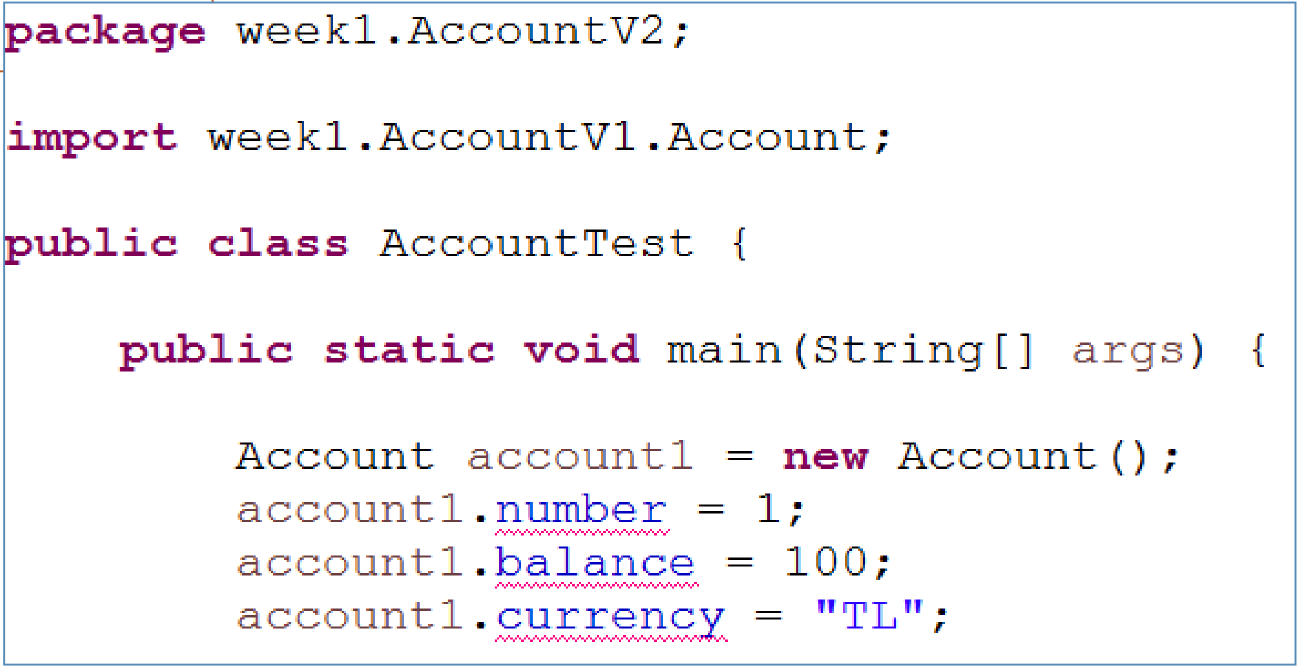
- Let’s try to use them from outside the package



# Access Specification



```
package week1.AccountV1;

public class Account {
    int number;
    double balance;
    String currency;
}
```

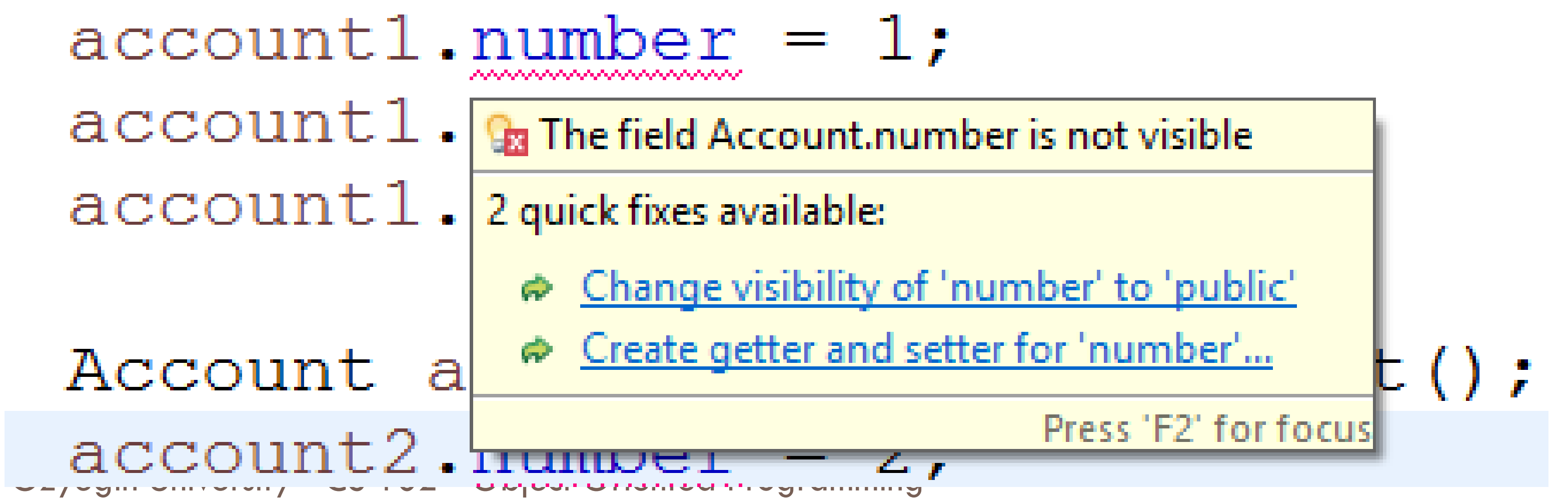


```
package week1.AccountV2;

import week1.AccountV1.Account;

public class AccountTest {

    public static void main(String[] args) {

        Account account1 = new Account();
        account1.number = 1;
        account1.balance = 100;
        account1.currency = "TL";
```

# Access Specification

```
package week1.AccountV1;

public class Account {
    int number;
    double balance;
    String currency;
}
```

```
package week1.AccountV2;

import week1.AccountV1.Account;
```



```
account1.number = 1;
account1.
account1.

Account a                                    t();
account2.number = 2;
```

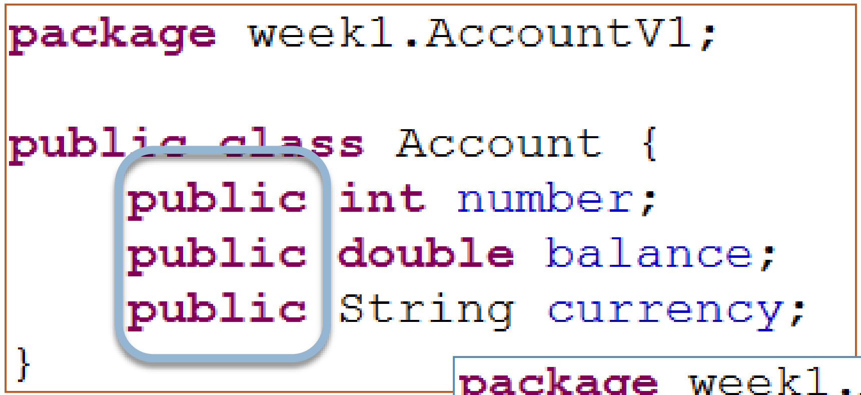
The field Account.number is not visible

2 quick fixes available:

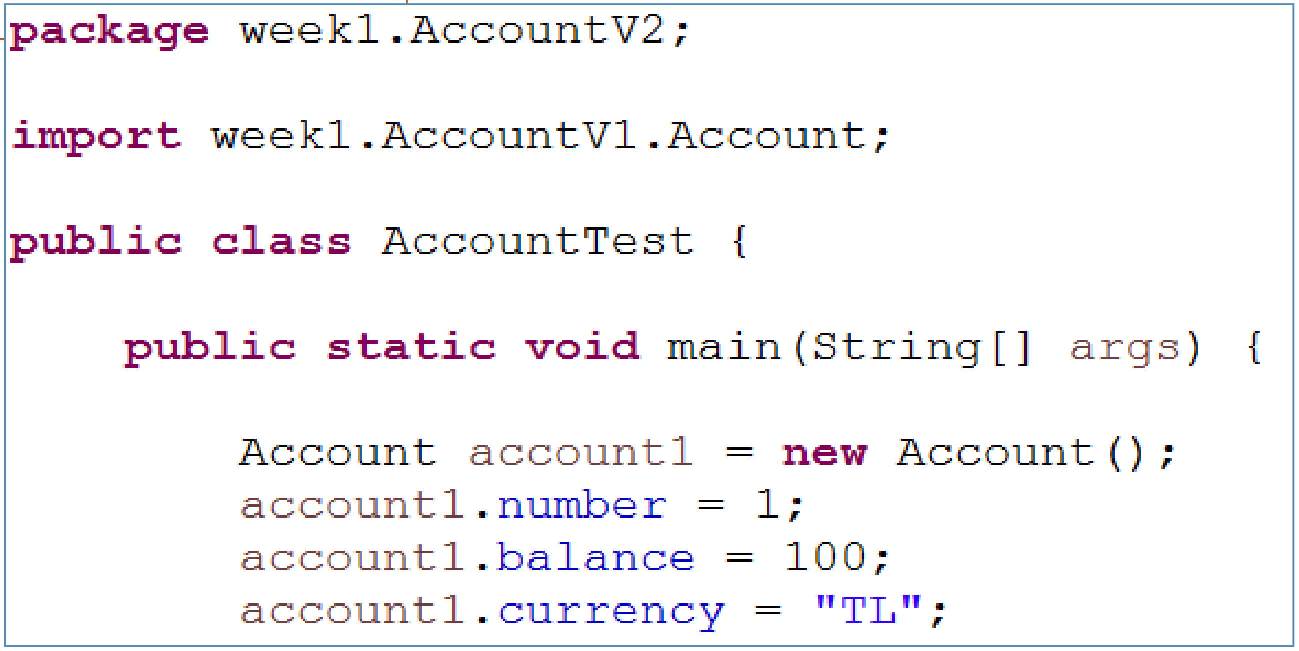
- Change visibility of 'number' to 'public'
- Create getter and setter for 'number'...

Press 'F2' for focus

# Access Specification



```
package week1.AccountV1;

public class Account {
    public int number;
    public double balance;
    public String currency;
}
```



```
package week1.AccountV2;

import week1.AccountV1.Account;

public class AccountTest {

    public static void main(String[] args) {

        Account account1 = new Account();
        account1.number = 1;
        account1.balance = 100;
        account1.currency = "TL";
```

## Access Specification

```
package week1.AccountV1;

public class Account {
    public int number;
    public double balance;
    public String currency;
}
```

**number, balance** and **currency** are visible in everywhere!

```
package week1.AccountV2;

import week1.AccountV1.Account;

public class AccountTest {

    public static void main(String[] args) {

        Account account1 = new Account();
        account1.number = 1;
        account1.balance = 100;
        account1.currency = "TL";
```

No access related errors!

## Important!!!

- However, making everything **public is not the solution.**
  - When something is public, it can be accessed and also can be modified from everywhere!
- It is also **not a good idea to leave it package-private.**
  - In default case (without any access modifier) that information can be accessed and modified everywhere within the package.
- These are not optimum solutions.
- You should encapsulate that information and limit its access and make sure that it can be modified only within your control.

## Controlling Access to Entries

- Each entry (class, class instance, member function) in a Java class is marked with one of the following keywords to control which classes have access to that entry:
  - public
  - private
  - no keyword (default)
  - protected

## Controlling Access to Entries

- Each entry (class, class instance, member function) in a Java class is marked with one of the following keywords to control which classes have access to that entry:
  - **public**: the entry is accessible from everywhere
  - private
  - no keyword (default)
  - protected

## Controlling Access to Entries

- Each entry (class, class instance, member function) in a Java class is marked with one of the following keywords to control which classes have access to that entry:
  - public
  - **private**: the entry is accessible only within the class, invisible everywhere outside the class
  - no keyword (default)
  - protected

## Controlling Access to Entries

- Each entry (class, class instance, member function) in a Java class is marked with one of the following keywords to control which classes have access to that entry:
  - public
  - private
  - **no keyword (default)**: entry is accessible to classes inside the same package, invisible to all the others. **package private.**
  - protected

# Controlling Access to Entries

- Each entry (class, class instance, member function) in a Java class is marked with one of the following keywords to control which classes have access to that entry:
  - public
  - private
  - no keyword (default)
  - **protected**: entry is accessible to the class itself, other classes inside the same package and all subclasses.

## Access Modifiers

- Which one is the most restrictive one?
  - public
  - private
  - no keyword (default)
  - protected

# Access Modifiers

- Which one is the most restrictive one?
  - public
  - **private**
  - no keyword (default)
  - protected

## Access Modifiers

- Which one is the least restrictive one?
  - public
  - private
  - no keyword (default)
  - protected

## Access Modifiers

- Which one is the least restrictive one?
  - **public**
  - private
  - no keyword (default)
  - protected

## Access Modifiers

- Rank them in increasing order of restrictiveness?
  - public
  - private
  - no keyword (default)
  - protected

## Access Modifiers

- Rank them in increasing order of restrictiveness?
  - public
  - private
  - no keyword (default)
  - protected
- Answer:
  - public, protected, default, private
    - **protected** entities can be accessed by subclasses in other packages

## Access Modifiers: Access levels

- private: the class itself
- default: private + classes inside the same package
- protected: default + all subclasses
- public: all classes

## Access Modifiers: Access levels

| | Class | Package | Subclass | World |
|---|---|---|---|---|
| public | Y | | | |
| protected | | | | |
| default | | | | |
| private | | | | N |

Source: http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

## Access Modifiers: Access levels

| | Class | Package | Subclass | World |
|---|---|---|---|---|
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| default | Y | Y | N | N |
| private | Y | N | N | N |

Source: http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

## Important!!!

- However, making everything **public is not the solution.**
  - When something is public, it can be accessed and also can be modified from everywhere!
- It is also **not a good idea to leave it package-private.**
  - In default case (without any access modifier)  that information can be accessed and modified everywhere within the package.
- These are not optimum solutions.
- You should encapsulate that information and limit its access and make sure that it can be modified only within your control.

# For most of the cases…

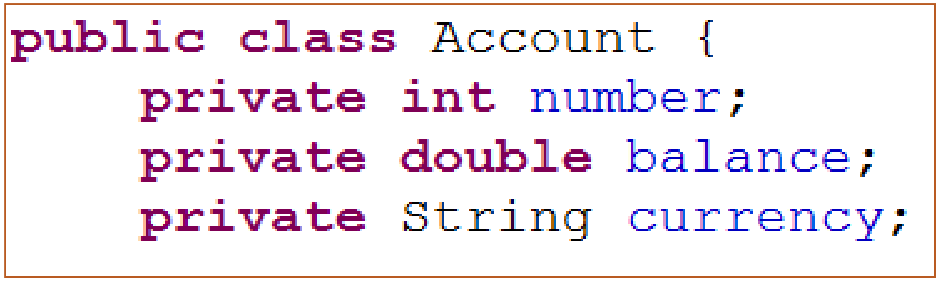
- **Class instances** should be **private**
  - Only the class itself can access these variables
  - They are visible only inside the class definition
    - Only member functions of the class can access them
  - They are invisible outside the class
  - Therefore, the control is on the class itself only.
- There may be times for exceptions.
  - Example: during inheritance

## For most of the cases...

- **Class methods** should be **public** or **private**
  - **public** if they will be used publicly
  - **private** if they are useful for another class function but not to be used by other classes directly
- There can be exceptions to these.

## Bank Account – version 10

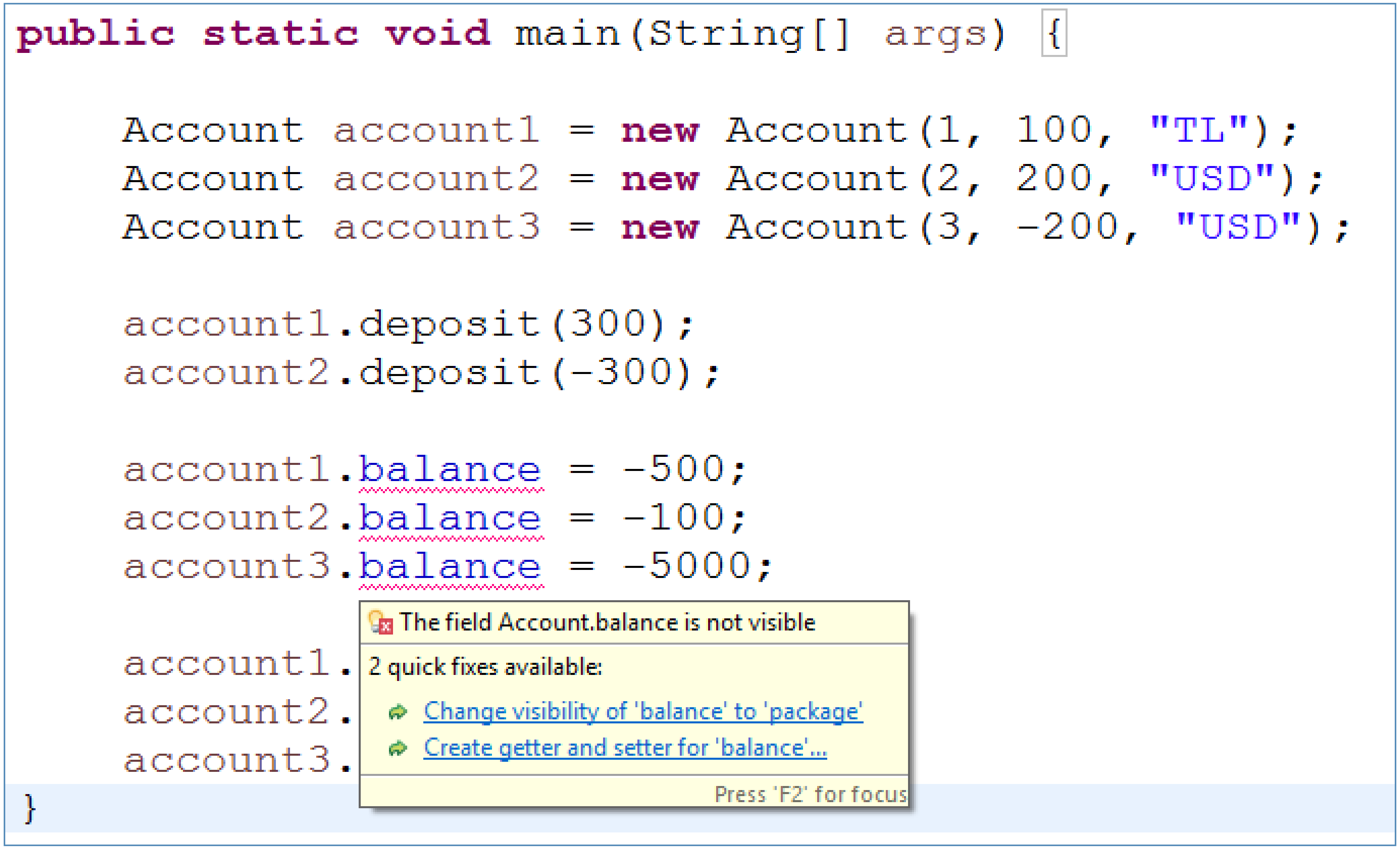
- Class instances



```
public class Account {
    private int number;
    private double balance;
    private String currency;
```

- Member functions were public already

## No write access



```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");
    Account account3 = new Account(3, -200, "USD");

    account1.deposit(300);
    account2.deposit(-300);

    account1.balance = -500;
    account2.balance = -100;
    account3.balance = -5000;

    account1.
    account2.
    account3.
}
```

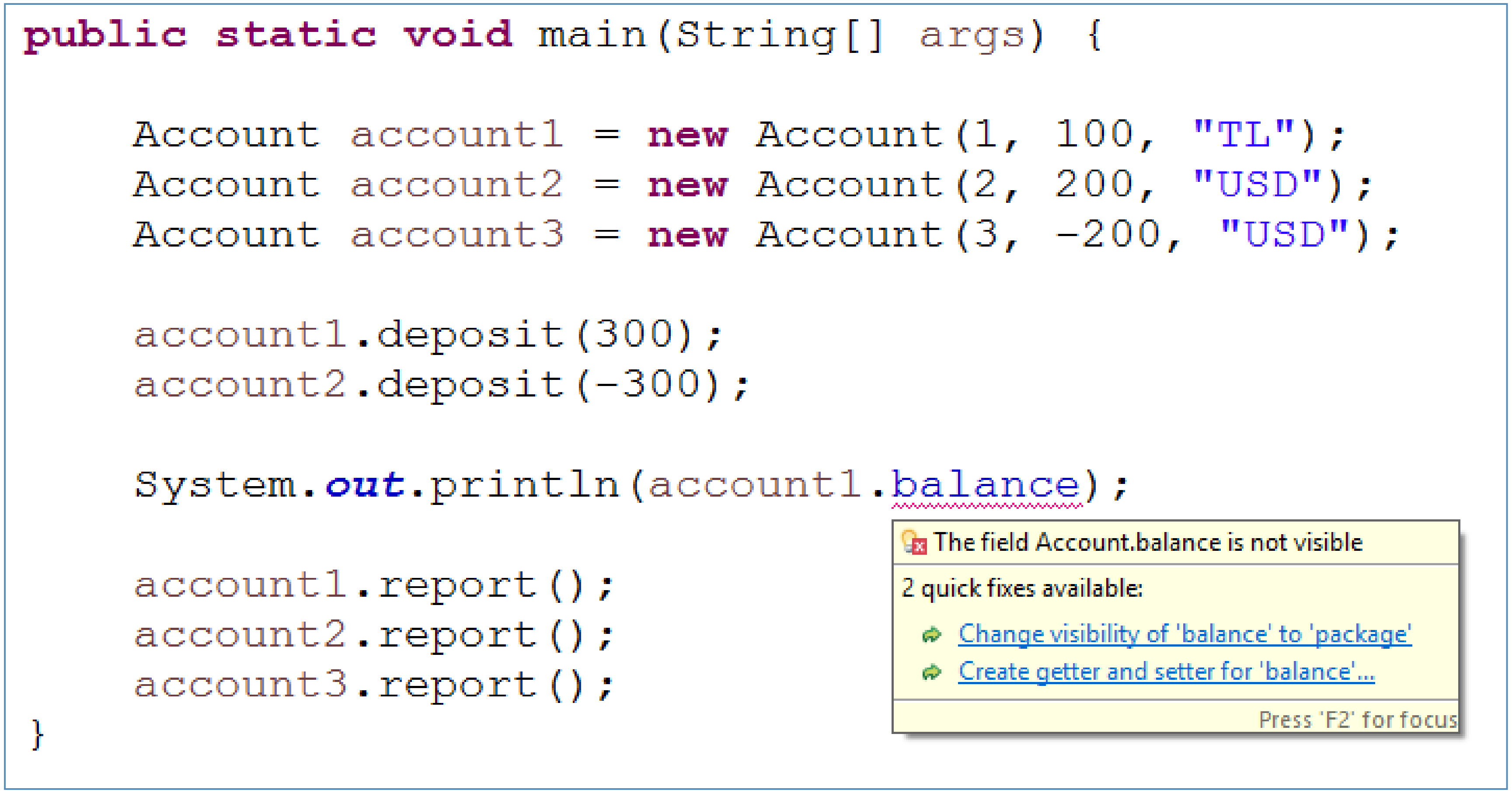
The field Account.balance is not visible

2 quick fixes available:

- Change visibility of 'balance' to 'package'
- Create getter and setter for 'balance'...

Press 'F2' for focus

# No read access



```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");
    Account account3 = new Account(3, -200, "USD");

    account1.deposit(300);
    account2.deposit(-300);

    System.out.println(account1.balance);

    account1.report();
    account2.report();
    account3.report();
}
```

The field Account.balance is not visible

2 quick fixes available:

- Change visibility of 'balance' to 'package'
- Create getter and setter for 'balance'...

Press 'F2' for focus

## Accessing Class Instances

- Since class instances are private, we won’t have direct access to those instances
  - no read or write access
- How can we access them?

## Accessing Class Instances

- Since class instances are private, we won't have direct access to those instances
  - no read or write access
- How can we access them?
  - by using getters and setters

# Getters and Setters

- Get and set methods allow customized access to class instances
  - **getter** for **read** access
    - returns the class instance without modifying
  - **setter** for **write** access
    - modifies the class instance
    - mostly assigns the function argument’s value to the class instance

## An example getter function

- **getter** for **read** access
  - returns the class instance without modifying

```
public int getNumber()   {
        return number;
}
```

## An example getter function

- **getter** for **read** access
  - returns the class instance without modifying

```
public int getNumber()   {
      return number;
}
```

  - What other getter functions do we need?

## Getter Function

```
public class Account {
    private int number;
    private double balance;
    private String currency;
```

```
public int getNumber()   {
    return number;
}
public double getBalance()   {
    return balance;
}
public String getCurrency(){
    return currency;
}
```

## Setters

- Using private for class instances give more control to the class.
- The class can enforce legal value assignments through setters.

## An example setter function

- **setter** for **write** access
  - modifies the class instance
  - mostly assigns the function argument's value to the class instance

```
public void setCurrency(String c)    {
        currency = c;
}
```

# An example setter function

- **setter** for **write** access
  - modifies the class instance
  - mostly assigns the function argument's value to the class instance

```
public void setCurrency(String c)    {
        currency = c;
 }
```

  - Do we need other setter functions?

## Setter Functions

- Do we need other setter functions?
  - account number
    - Initialized when an account is created
    - Cannot be changed afterwards
  - account balance
    - We don’t use a set function but instead
      - Deposit: to put money in a bank account
      - Withdraw: to remove money from a bank account

Source:http://www.merriam-webster.com/dictionary

## deposit and withdraw functions

- We already have the deposit function

```
public void deposit(double d) {
    if (d > 0)  {
        balance = balance + d;
        System.out.println(
                d + " " + currency + " have been deposited");
        System.out.println(
                "The balance is " + balance + " " + currency);
    }
    else
        System.out.println(
                "The amount should be positive!");
}
```

# deposit and withdraw functions

- Can you write down the withdraw function?

# deposit and withdraw functions

- Can you write down the withdraw function?
  - Do not let withdraw if
    - withdraw amount is negative
    - withdraw amount is larger than the balance
  - Otherwise
    - withdraw the money and update the balance

# deposit and withdraw functions

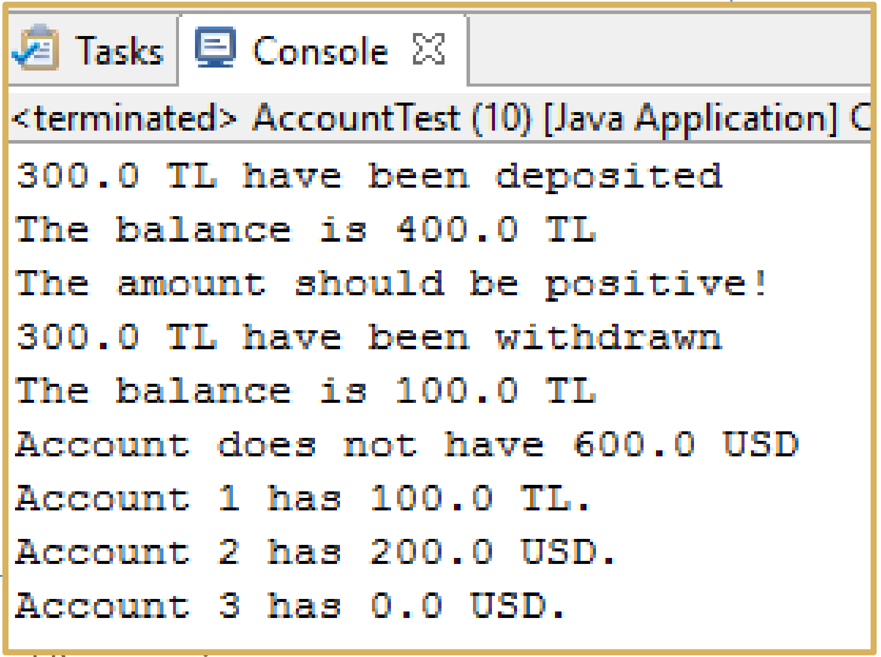
```
public void withdraw(double d) {
    if (d > 0)  {
        if (balance < d)    {
            System.out.println(
                    "Account does not have " + d + " " + currency);
        }
        else {
            balance = balance - d;
            System.out.println(
                    d + " " + currency + " have been withdrawn");
            System.out.println(
                    "The balance is " + balance + " " + currency);
        }
    }
    else
        System.out.println(
                "The amount should be positive!");
}
```

# deposit and withdraw functions

```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");
    Account account3 = new Account(3, -200, "USD");

    account1.deposit(300);
    account2.deposit(-300);

    account1.withdraw(300);
    account2.withdraw(600);

    account1.report();
    account2.report();
    account3.report();
}
```

# deposit and withdraw functions

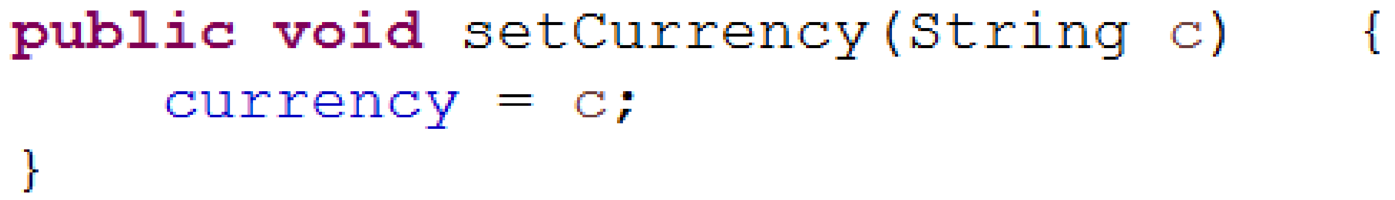
```java
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");
    Account account3 = new Account(3, -200, "USD");

    account1.deposit(300);
    account2.deposit(-300);

    account1.withdraw(300);
    account2.withdraw(600);

    account1.report();
    account2.report();
    account3.report();
}
```



```
<terminated> AccountTest (10) [Java Application] C
300.0 TL have been deposited
The balance is 400.0 TL
The amount should be positive!
300.0 TL have been withdrawn
The balance is 100.0 TL
Account does not have 600.0 USD
Account 1 has 100.0 TL.
Account 2 has 200.0 USD.
Account 3 has 0.0 USD.
```

# setCurrency function

- Lets review setCurrency function



```
public void setCurrency(String c)    {
        currency = c;
 }
```

- 1 USD = 2.9 TL
- How should we modify the above function?

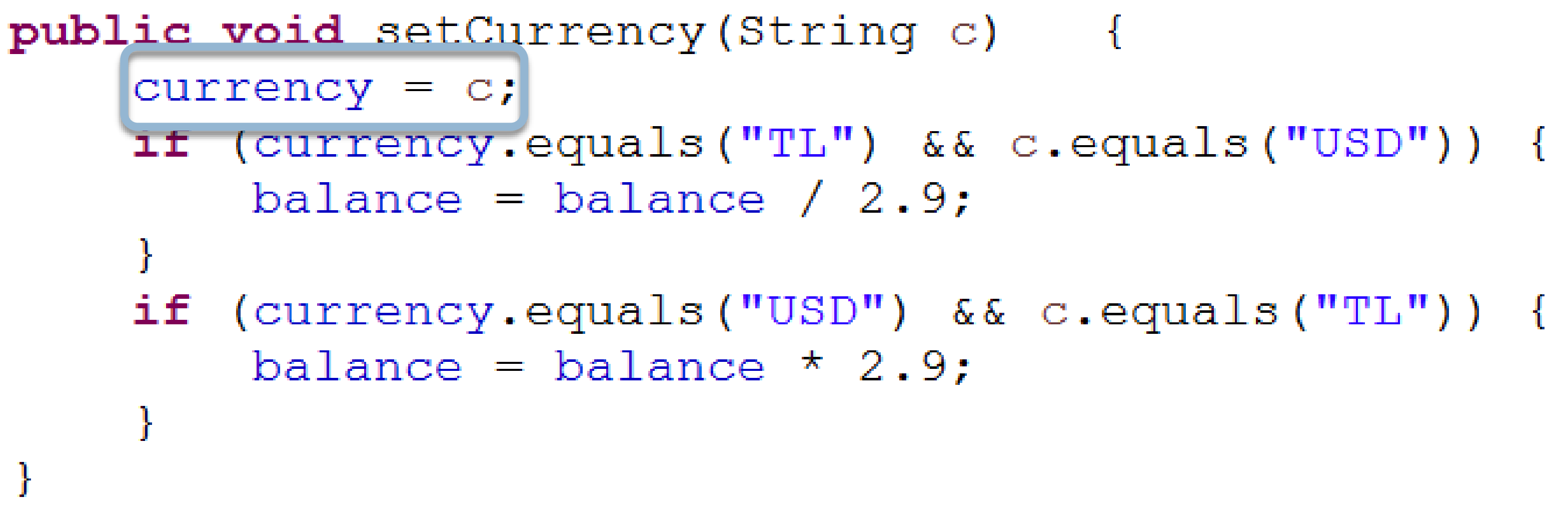
## setCurrency function

- Will this work?

```
public void setCurrency(String c)    {
     currency = c;
     if (currency.equals("TL") && c.equals("USD")) {
          balance = balance / 2.9;
     }
     if (currency.equals("USD") && c.equals("TL")) {
          balance = balance * 2.9;
     }
}
```

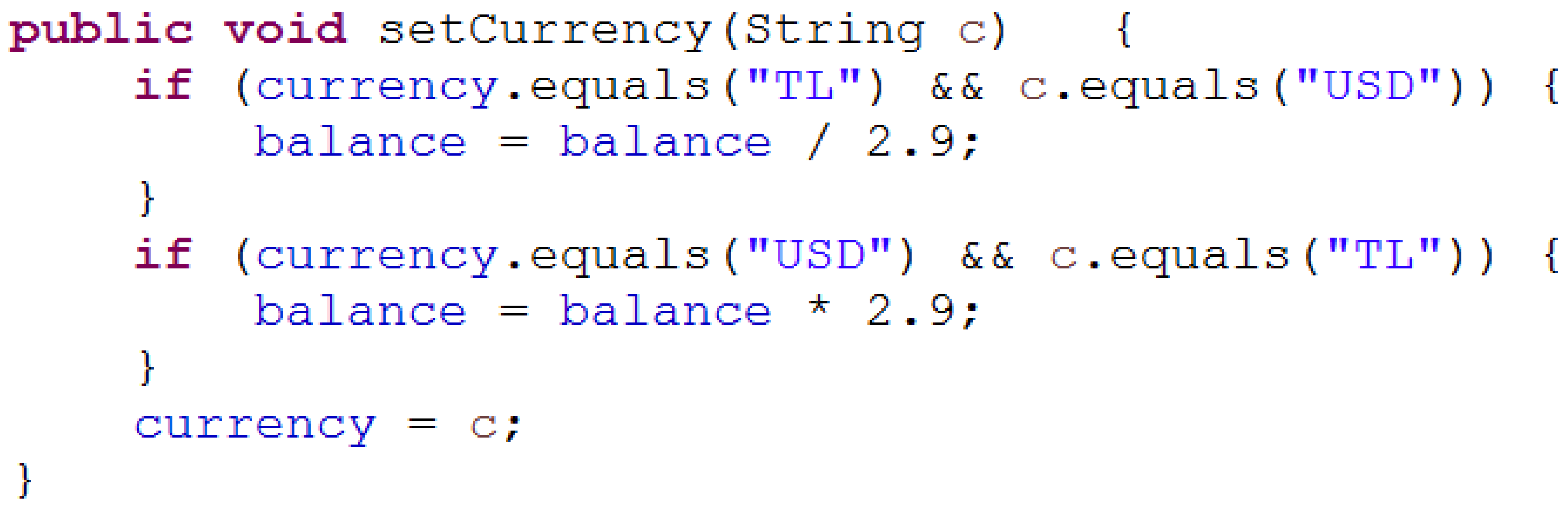
## setCurrency function

- Will this work?



```
public void setCurrency(String c)    {
    currency = c;
    if (currency.equals("TL") && c.equals("USD")) {
        balance = balance / 2.9;
    }
    if (currency.equals("USD") && c.equals("TL")) {
        balance = balance * 2.9;
    }
}
```

# setCurrency function



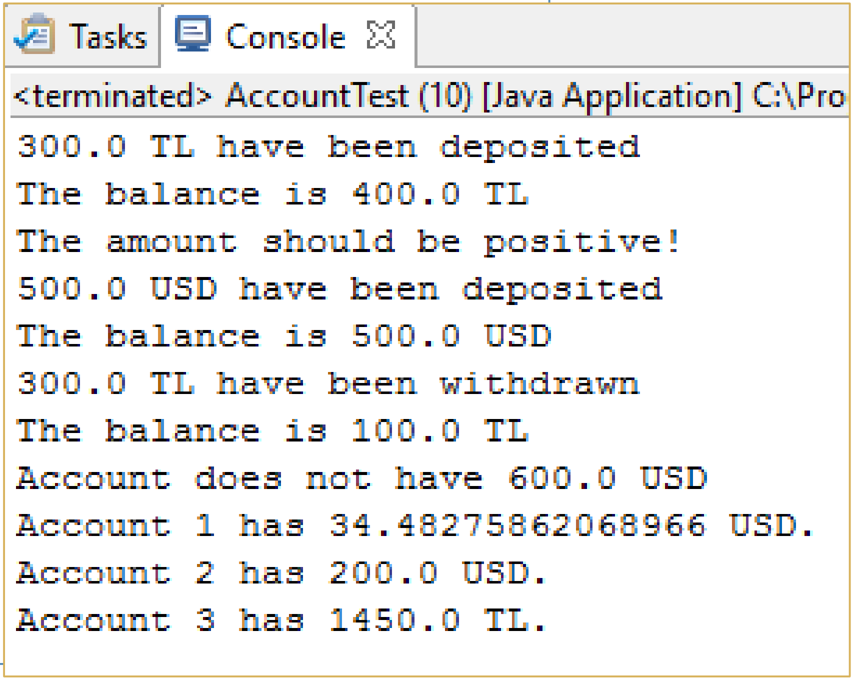
```
public void setCurrency(String c)    {
    if (currency.equals("TL") && c.equals("USD")) {
        balance = balance / 2.9;
    }
    if (currency.equals("USD") && c.equals("TL")) {
        balance = balance * 2.9;
    }
    currency = c;
}
```

## What is the output?

```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");
    Account account3 = new Account(3, -200, "USD");

    account1.deposit(300);
    account2.deposit(-300);
    account3.deposit(500);

    account1.withdraw(300);
    account2.withdraw(600);

    account3.setCurrency("TL");
    account1.setCurrency("USD");

    account1.report();
    account2.report();
    account3.report();
}
```

# What is the output?

```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");
    Account account3 = new Account(3, -200, "USD");

    account1.deposit(300);
    account2.deposit(-300);
    account3.deposit(500);

    account1.withdraw(300);
    account2.withdraw(600);

    account3.setCurrency("TL");
    account1.setCurrency("USD");

    account1.report();
    account2.report();
    account3.report();
}
```



```
<terminated> AccountTest (10) [Java Application] C:\Pro
300.0 TL have been deposited
The balance is 400.0 TL
The amount should be positive!
500.0 USD have been deposited
The balance is 500.0 USD
300.0 TL have been withdrawn
The balance is 100.0 TL
Account does not have 600.0 USD
Account 1 has 34.48275862068966 USD.
Account 2 has 200.0 USD.
Account 3 has 1450.0 TL.
```

## Unknown currency?

- What happens in the following case?

```
account3.setCurrency("TL");
account1.setCurrency("USD");
account2.setCurrency("AKCE");
```

```
public void setCurrency(String c)    {
    if (currency.equals("TL") && c.equals("USD")) {
        balance = balance / 2.9;
    }
    if (currency.equals("USD") && c.equals("TL")) {
        balance = balance * 2.9;
    }
    currency = c;
}
```

## Unknown currency?

- How can we fix this setCurrency function?

## Fixing setCurrency Function

```
public void setCurrency(String c)    {
    if (currency.equals("TL") && c.equals("USD")) {
        balance = balance / 2.9;
    }
    if (currency.equals("USD") && c.equals("TL")) {
        balance = balance * 2.9;
    }
    currency = c;
}
```

```
public void setCurrency(String c)    {
    if (currency.equals("TL") && c.equals("USD")) {
        balance = balance / 2.9;
    }
    if (currency.equals("USD") && c.equals("TL")) {
        balance = balance * 2.9;
    }
    if (c.equals("TL") || c.equals("USD")) {
        currency = c;
    }
}
```

# Unknown currency?

- The same thing can happen in constructor as well.

```
// Constructors
public Account(int n, double b, String c) {
    number = n;
    if (b > 0)
        balance = b;
    else
        balance = 0;
    currency = c;
}
public Account(int n, String c) {
    number = n;
    balance = 0;
    currency = c;
}
public Account(int n) {
    number = n;
    balance = 0;
    currency = "TL";
}
```

## Unknown currency?

- The same thing can happen in constructor as well.
- In default we should set it to “TL”

# Fixing Constructors (Account V12)

```java
// Constructors
public Account(int n, double b, String c) {
    number = n;
    if (b > 0)
        balance = b;
    else
        balance = 0;
    currency = c;
}
public Account(int n, String c) {
    number = n;
    balance = 0;
    currency = c;
}
```

```java
// Constructors
public Account(int n, double b, String c) {
    number = n;
    if (b > 0)
        balance = b;
    else
        balance = 0;

    if (c.equals("USD"))
        currency = c;
    else
        currency = "TL";
}
public Account(int n, String c) {
    number = n;
    balance = 0;

    if (c.equals("USD"))
        currency = c;
    else
        currency = "TL";
}
```

# Code Repetition

```java
// Constructors
public Account(int n, double b, String c) {
    number = n;
    if (b > 0)
        balance = b;
    else
        balance = 0;

    if (c.equals("USD"))
        currency = c;
    else
        currency = "TL";
}
public Account(int n, String c) {
    number = n;
    balance = 0;

    if (c.equals("USD"))
        currency = c;
    else
        currency = "TL";
}
```

# Code Repetition

```
// Constructors
public Account(int n, double b, String c) {
    number = n;
    if (b > 0)
        balance = b;
    else
        balance = 0;

    if (c.equals("USD"))
        currency = c;
    else
        currency = "TL";
}
public Account(int n, String c) {
    number = n;
    balance = 0;

    if (c.equals("USD"))
        currency = c;
    else
        currency = "TL";
}
```

How can we write a function for this check?

## Private Function

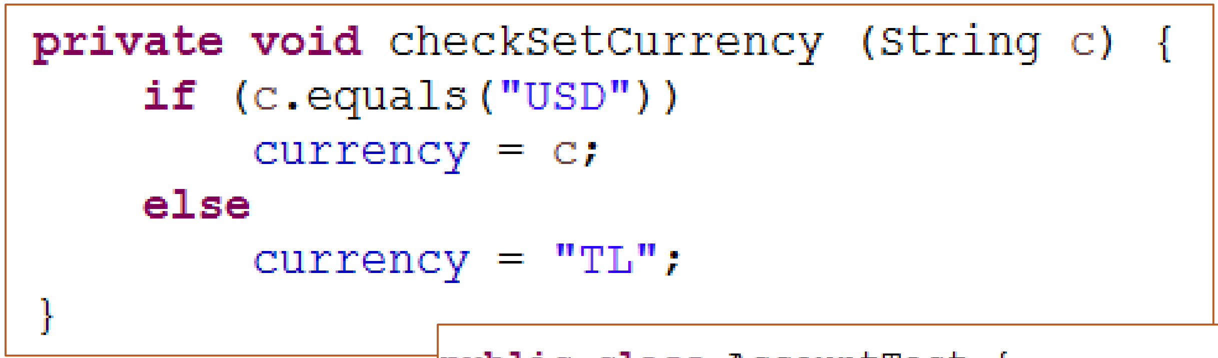
```
private void checkSetCurrency (String c) {
    if (c.equals("USD"))
        currency = c;
    else
        currency = "TL";
}
```

# Private Function

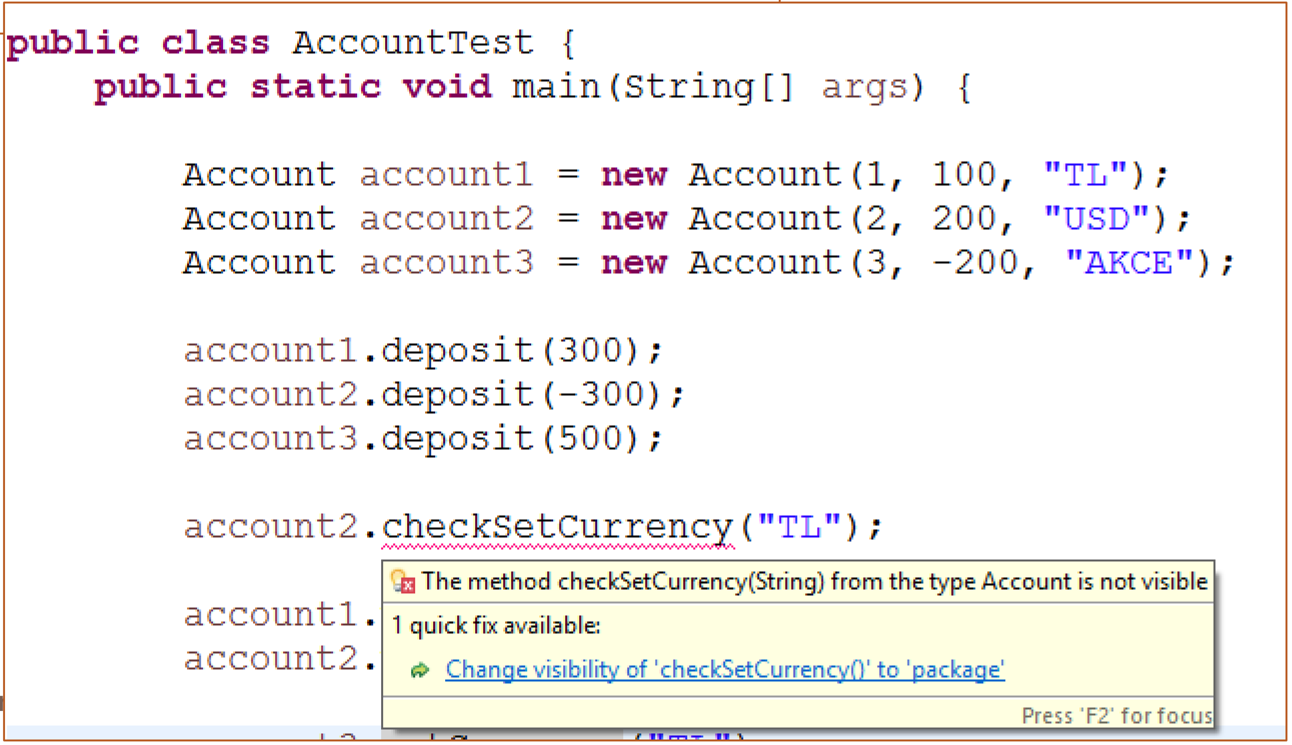
```java
private void checkSetCurrency (String c) {
    if (c.equals("USD"))
        currency = c;
    else
        currency = "TL";
}
```

```java
// Constructors
public Account(int n, double b, String c) {
    number = n;
    if (b > 0)
        balance = b;
    else
        balance = 0;

    checkSetCurrency(c);
}
public Account(int n, String c) {
    number = n;
    balance = 0;

    checkSetCurrency(c);
}
```

# Private Function



```
private void checkSetCurrency (String c) {
    if (c.equals("USD"))
        currency = c;
    else
        currency = "TL";
}
```



```
public class AccountTest {
    public static void main(String[] args) {

        Account account1 = new Account(1, 100, "TL");
        Account account2 = new Account(2, 200, "USD");
        Account account3 = new Account(3, -200, "AKCE");

        account1.deposit(300);
        account2.deposit(-300);
        account3.deposit(500);

        account2.checkSetCurrency("TL");

        account1.
        account2.
```

The method checkSetCurrency(String) from the type Account is not visible

1 quick fix available:

Change visibility of 'checkSetCurrency()' to 'package'

Press 'F2' for focus

## Private Function

- Functions which are helper functions to other member functions should be kept private.
  - Private function can be accessed from within the class.
  - Private function can not be accessed from outside the class.

# Get and Set Functions

- Setter methods usually begins with **‘set’** prefix.
  - setCurrency
- Getter methods usually begins with **‘get’** prefix.
  - getCurrency
  - But there is an exception for Boolean values
    - For Boolean values the prefix **‘is’** usually used.

## Boolean Get Functions (Account V13)

- Getter methods usually begins with **‘get’** prefix.
  - But there is an exception for Boolean values
    - For Boolean values the prefix **‘is’** usually used.

  - Assume that some accounts can be **active** while some of them are not. They can be **on hold.**
    - Keep active information within a boolean

## Boolean Get Functions (Account V13)

- Getter methods usually begins with '**get**' prefix.
  - But there is an exception for Boolean values
    - For Boolean values the prefix '**is**' usually used.

```
private int number;
private double balance;
private String currency;
private boolean active;

// Constructors
public Account(int n, double b, String c) {
    number = n;
    if (b > 0)
        balance = b;
    else
        balance = 0;

    checkSetCurrency(c);
    active = true;
}
```

# Get Functions (Account V13)

```
public int getNumber()   {
    return number;
}
public double getBalance()   {
    return balance;
}
public String getCurrency() {
    return currency;
}
public boolean isActive()    {
    return active;
}
```

# Get Functions (Account V13)

- For set functions you can still use ‘set’ prefix
  - setActive

```
public void setActive(boolean a)     {
     active = a;
}
```

```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");

    account1.setActive(false);
    System.out.println(account1.isActive());

    System.out.println(account2.isActive());
}
```

## Get Functions (Account V13)

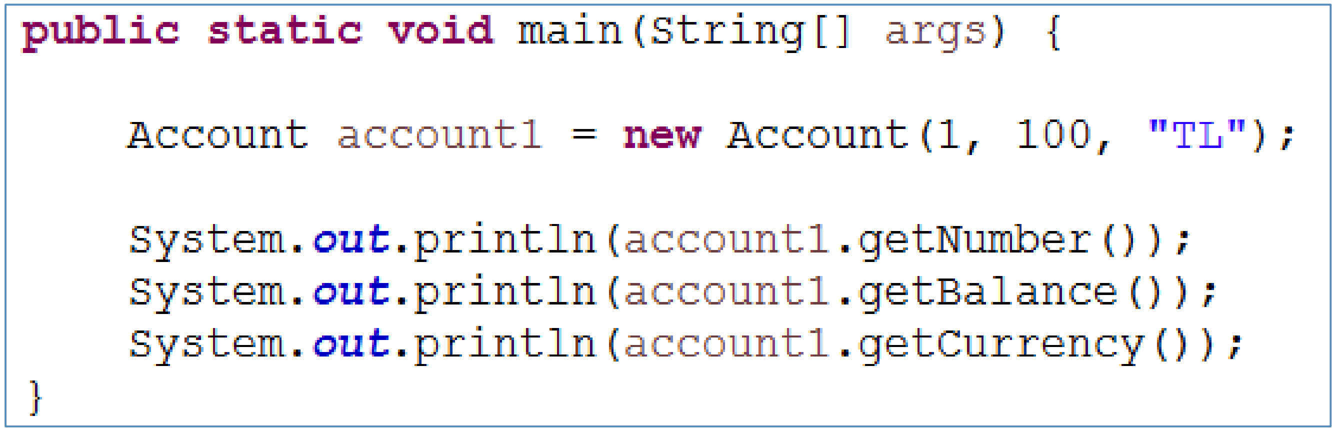
- For set functions you can still use ‘set’ prefix
  - setActive

```
public void setActive(boolean a)     {
     active = a;
}
```

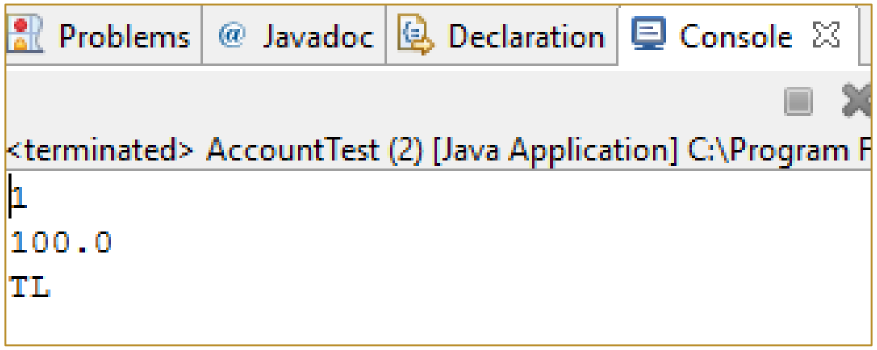
```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");
    Account account2 = new Account(2, 200, "USD");

    account1.setActive(false);
    System.out.println(account1.isActive());

    System.out.println(account2.isActive());
}
```

## Ways of printing out the object - 1

- get methods for accessing class instances one by one



```java
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");

    System.out.println(account1.getNumber());
    System.out.println(account1.getBalance());
    System.out.println(account1.getCurrency());
}
```



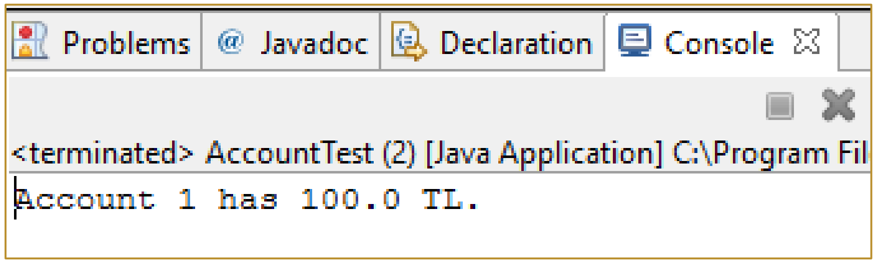
```
<terminated> AccountTest (2) [Java Application] C:\Program F
1
100.0
TL
```

# Ways of printing out the object - 2

- report method for printing report of the account

```
public void report() {
    System.out.println("Account " + number
            + " has " + balance
            + " " + currency + ".");
}
```

```
public static void main(String[] args) {

    Account account1 = new Account(1, 100, "TL");

    account1.report();
}
```



```
<terminated> AccountTest (2) [Java Application] C:\Program Fil
Account 1 has 100.0 TL.
```

# Ways of printing out the object

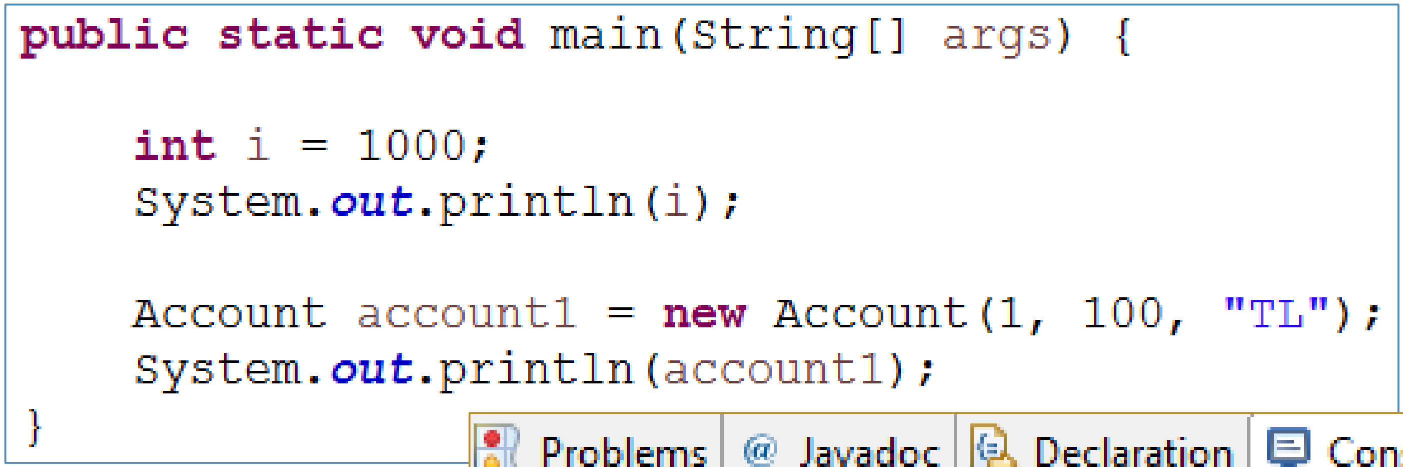
- Similar to other primitive types, can we just use the object inside System.out.println() function?

```
public static void main(String[] args) {

    int i = 1000;
    System.out.println(i);

    Account account1 = new Account(1, 100, "TL");
    System.out.println(account1);
}
```

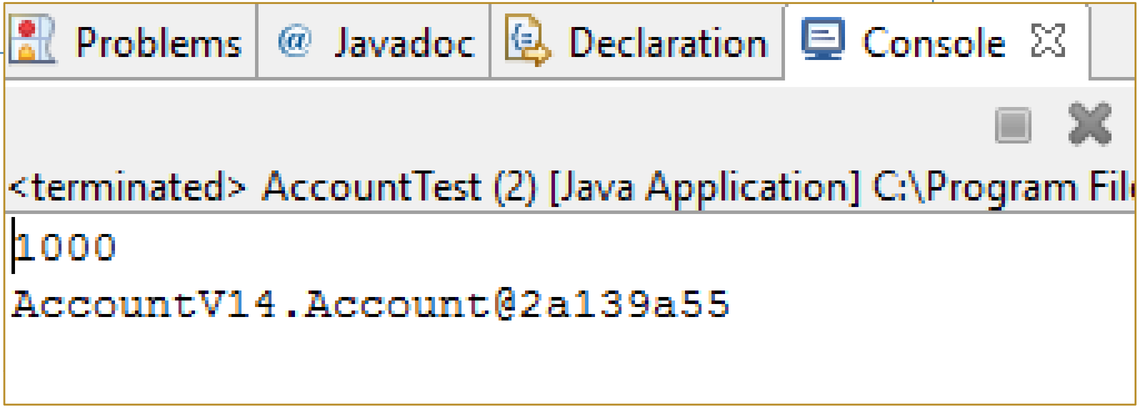
- What do you think the output will look like?

# Ways of printing out the object

- Similar to other primitive types, can we just use the object inside System.out.println() function?



```
public static void main(String[] args) {

    int i = 1000;
    System.out.println(i);

    Account account1 = new Account(1, 100, "TL");
    System.out.println(account1);
}
```



```
<terminated> AccountTest (2) [Java Application] C:\Program Fil
1000
AccountV14.Account@2a139a55
```

# Ways of printing out the object

- Similar to other primitive types, can we just use the object inside System.out.println() function?

```
public static void main(String[] args) {

    int i = 1000;
    System.out.println(i);

    Account account1 = new Account(1, 100, "TL");
    System.out.println(account1);
}
```

- In order to get smt meaningful, we need to override **toString** method of the class.

# toString method

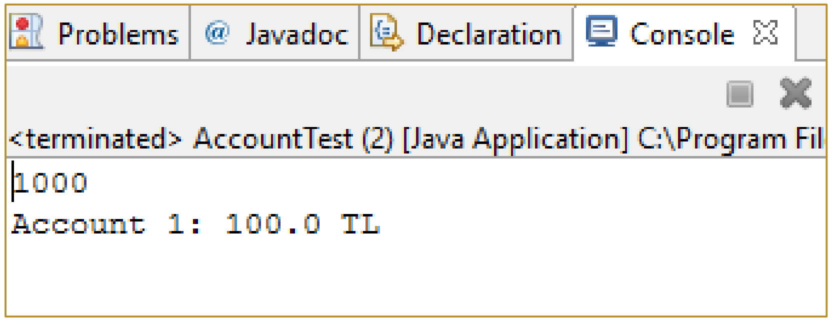
- toString method tells Java how to display an object of the class.
- It returns a String representation of the object.

```
public String toString() {
    return "Account " + number + ": " + balance + " " + currency;
}
```

## toString method

```
public String toString() {
    return "Account " + number + ": " + balance + " " + currency;
}
```

```
public static void main(String[] args) {

    int i = 1000;
    System.out.println(i);

    Account account1 = new Account(1, 100, "TL");
    System.out.println(account1);
}
```



```
<terminated> AccountTest (2) [Java Application] C:\Program Fil
1000
Account 1: 100.0 TL
```

## Announcements

- Midterm Dates
  - Midterm 1: 25 October 2016
  - Midterm 2: 6 December 2016

## Any Questions ?