ÖZYEĞİN
UNIVERSITY

CS105
Introduction to Object-Oriented Programming

Prof. Dr. Nizamettin AYDIN

naydin@itu.edu.tr

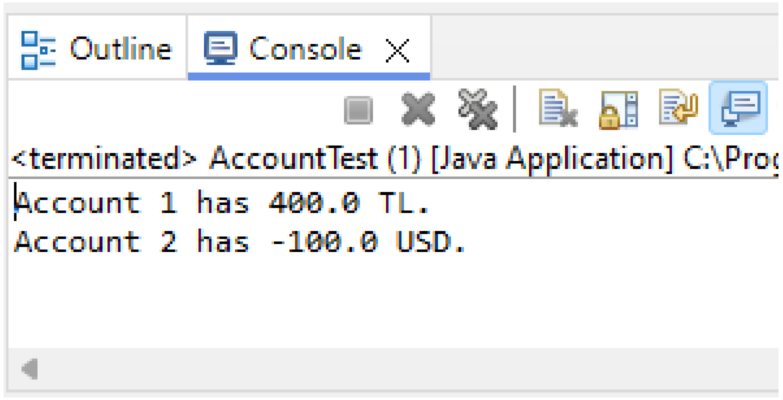
nizamettin.aydin@ozyegin.edu.tr

Access Modifiers

Outline

Bank Account – version 8

```
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 (1) [Java Application] C:\Prog
Account 1 has 400.0 TL.
Account 2 has -100.0 USD.
```

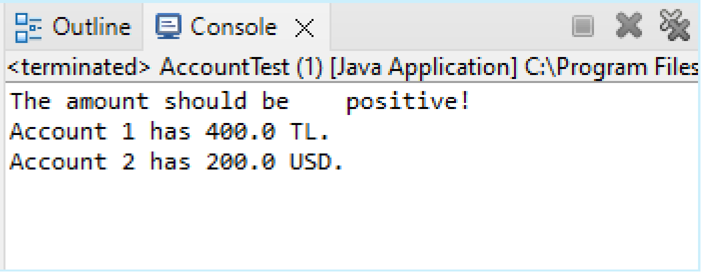
Definition of deposit

- Deposit
  - dɪˈpɒzɪt/
    - https://www.merriam-webster.com/dictionary/deposit
    - https://dictionary.cambridge.org/tr/s%C3%B6zl%C3%BCk/ingilizce/deposit
  - A sum of money placed in a bank account.
  - A payment made in advance, such as a security deposit for renting a property.
  - A layer of sediment that settles at the bottom of a liquid.
- We should not allow depositing negative amount of money.
- How?

deposit function

```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();
    }
}
```


Outline
Console
<terminated> AccountTest (1) [Java Application] C:\Program Files

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

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";
}
```

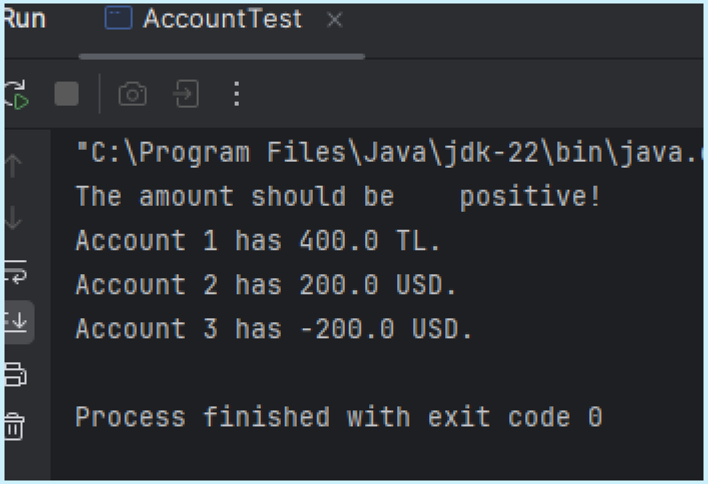
Constructors

- 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 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, "USD");

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

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



```
"C:\Program Files\Java\jdk-22\bin\java.
The amount should be    positive!
Account 1 has 400.0 TL.
Account 2 has 200.0 USD.
Account 3 has -200.0 USD.

Process finished with exit code 0
```

Constructors

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

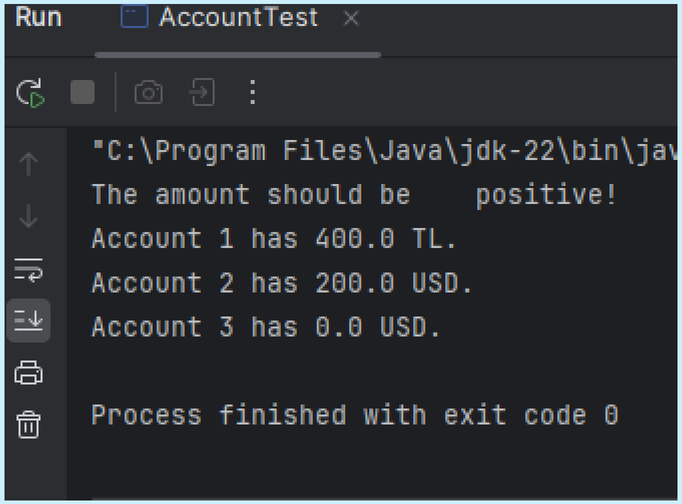
```
   public Account(int n, double b, String c){
      number = n;
      balance = b;
      currency = c;
   }
```

- 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 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, "USD");

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

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



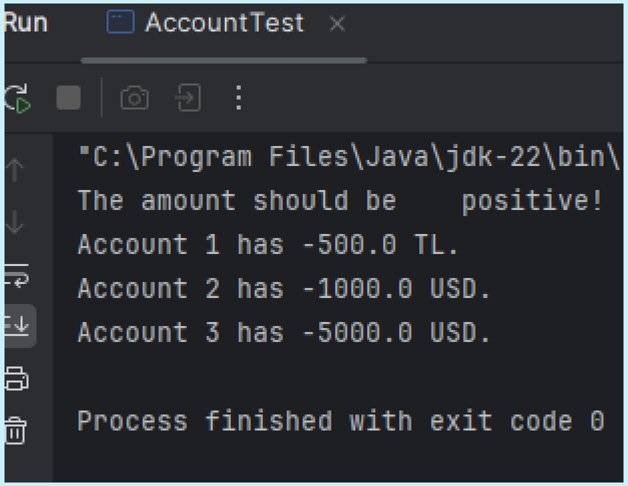
```
"C:\Program Files\Java\jdk-22\bin\jav
The amount should be    positive!
Account 1 has 400.0 TL.
Account 2 has 200.0 USD.
Account 3 has 0.0 USD.

Process finished with exit code 0
```

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?

```java
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, "USD");

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

        account1.balance = -500;
        account2.balance = -1000;
        account3.balance = -5000;

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



```
"C:\Program Files\Java\jdk-22\bin\
The amount should be    positive!
Account 1 has -500.0 TL.
Account 2 has -1000.0 USD.
Account 3 has -5000.0 USD.

Process finished with exit code 0
```

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 modifiers**
  - 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** :
    - visible to the earth

  or
  - **default** (no keyword) :
    - visible only within the same package

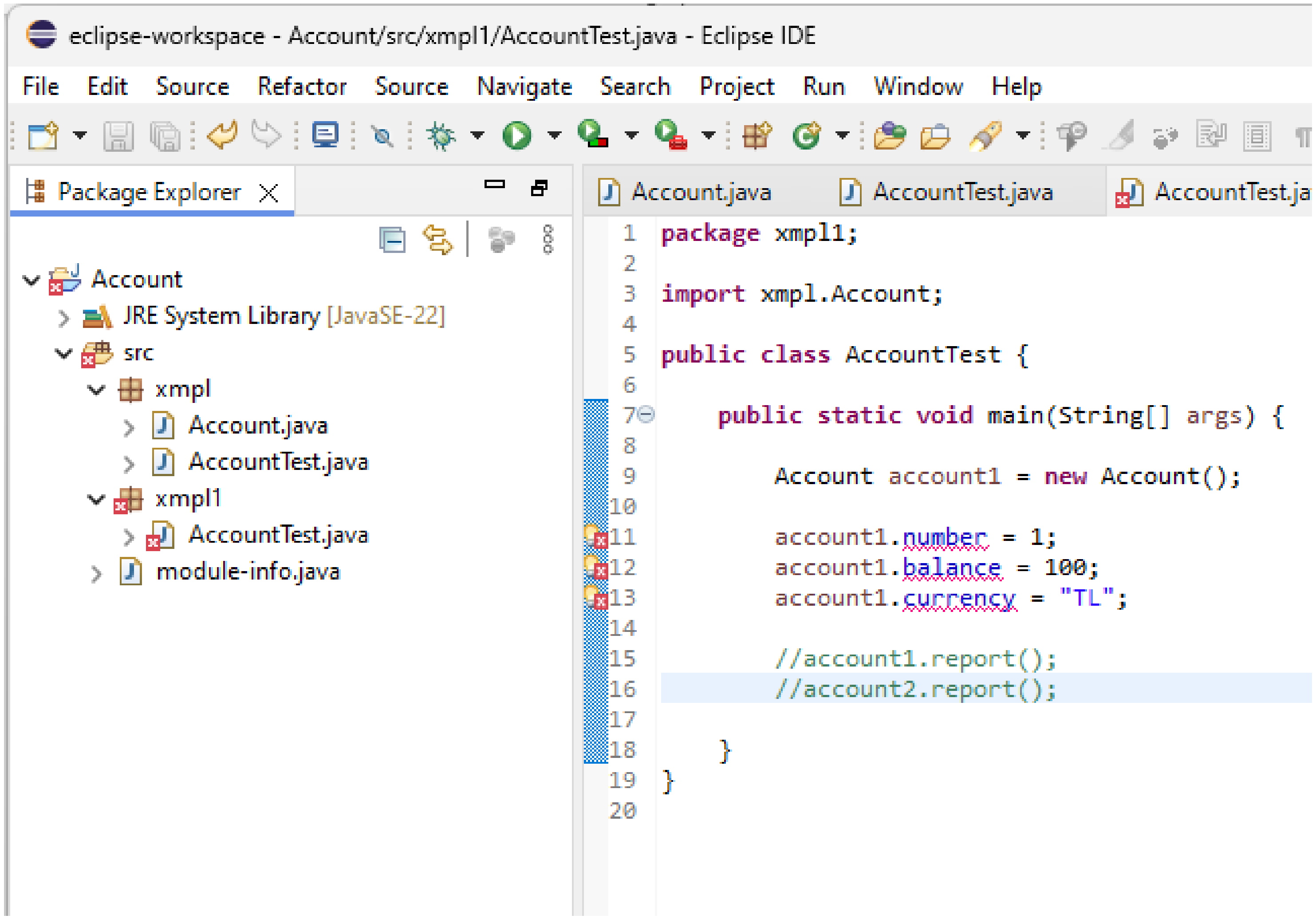
Access Specification

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

- These variables do not 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



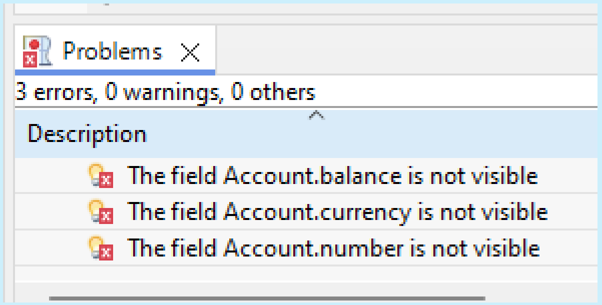
eclipse-workspace - Account/src/xmpl1/AccountTest.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Package Explorer
Account
JRE System Library [JavaSE-22]
src
xmpl
Account.java
AccountTest.java
xmpl1
AccountTest.java
module-info.java
Account.java
AccountTest.java
AccountTest.ja

```java
package xmpl1;

import xmpl.Account;

public class AccountTest {

    public static void main(String[] args) {

        Account account1 = new Account();

        account1.number = 1;
        account1.balance = 100;
        account1.currency = "TL";

        //account1.report();
        //account2.report();

    }
}

```

Access Specification

```
package xmpl;

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

```
package xmpl1;
import xmpl.Account;
public class AccountTest {
   public static void main(String[] args) {
      Account account1 = new Account();

      account1.number = 1;

      account1.balance = 100;

      account1.currency = "TL";

   }

}
```


Problems
3 errors, 0 warnings, 0 others

Description
The field Account.balance is not visible
The field Account.currency is not visible
The field Account.number is not visible

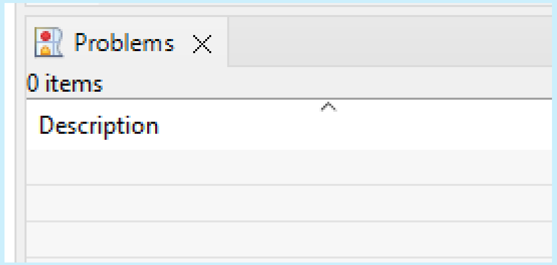
Access Specification

```
package xmpl;

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

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

```
package xmpl1;
import xmpl.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!



Problems
0 items
Description

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:
    - the entry is accessible from everywhere
  - private:
    - the entry is accessible only within the class, invisible everywhere outside the class
  - no keyword (default):
    - entry is accessible to classes inside the same package, invisible to all the others.
    - **package private**.
  - protected:
    - entry is accessible to the class itself, other classes inside the same package and all subclasses.

Access Modifiers

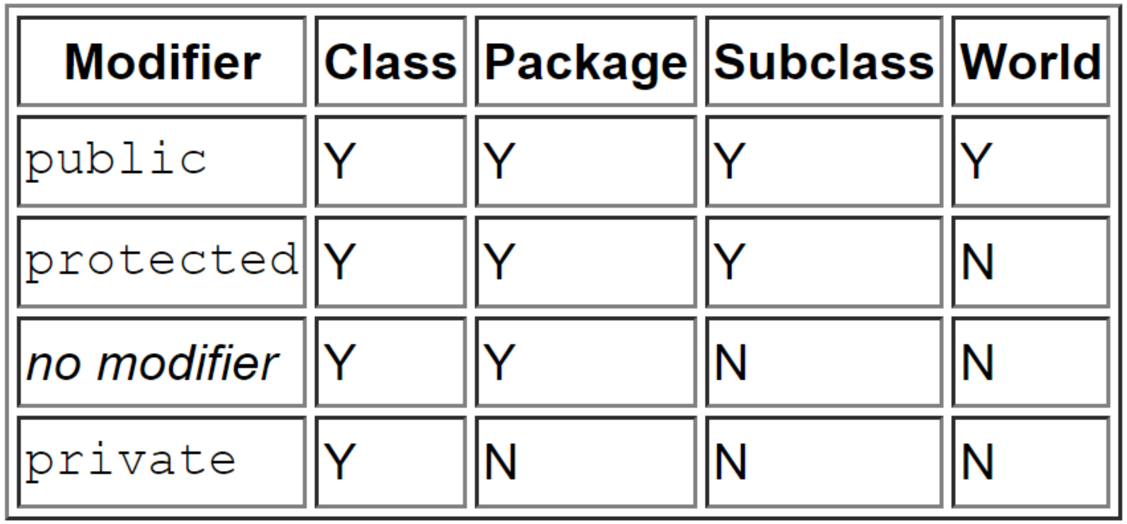
- Which one is the most restrictive one?
  - public
  - private
  - no keyword (default)
  - protected
- Which one is the least restrictive one?
  - public
  - private
  - no keyword (default)
  - protected
- Rank them in increasing order of restrictiveness?
  - public
  - private
  - no keyword (default)
  - protected

- **Answer:**
  - public, protected, default, private
    - **protected** entities can be accessed by sub-classes in other packages

Access Modifiers: Access levels

- private: the class itself
- default: private + classes inside the same package
- protected: default + all sub-classes
- public: all classes
- **the access to members permitted by each modifier:**

Access Levels



Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	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
- **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 9

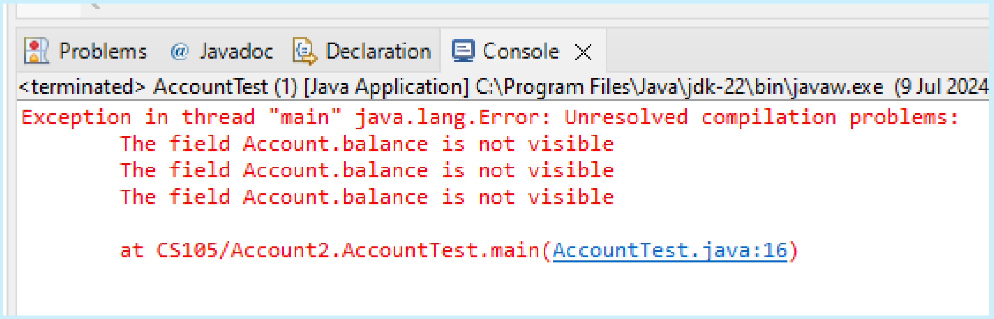
- Class instances

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

- Member functions were public already

No read/write access

```
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, "USD");

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

        account1.balance = -500;
        account2.balance = -1000;
        account3.balance = -5000;

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



```
<terminated> AccountTest (1) [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (9 Jul 2024
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
	The field Account.balance is not visible
	The field Account.balance is not visible
	The field Account.balance is not visible

	at CS105/Account2.AccountTest.main(AccountTest.java:16)
```

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**

- 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

Getters

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

- An example getter function:

```
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 int getBalance() {
     return balance;
}
public int getCurrency() {
     return currency;
}
```

Setters

- Using private for class instances gives more control to the class.
  - The class can enforce legal value assignments through setters.
- **setter** for write access
  - modifies the class instance
  - mostly assigns the function argument’s value to the class instance

- An example setter function:

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

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

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!");
}
```

- 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

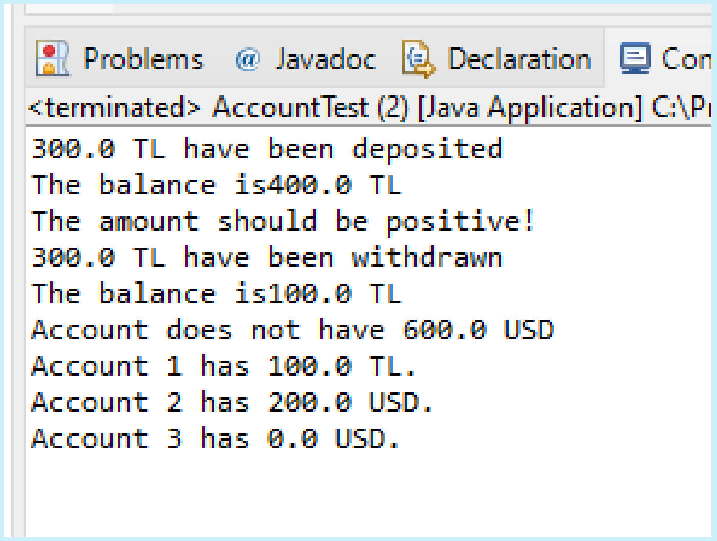
- Otherwise
  - withdraw the money and update the balance

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 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, "USD");

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

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

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



```
<terminated> AccountTest (2) [Java Application] C:\P
300.0 TL have been deposited
The balance is400.0 TL
The amount should be positive!
300.0 TL have been withdrawn
The balance is100.0 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

- Let's review setCurrency function

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

- 1 USD = 32.88 TL
- How should we modify the above function?
- Will this work?

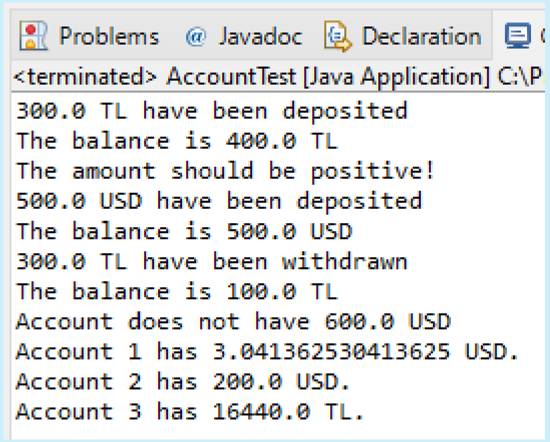
```
public void setCurrency(String c) {
    currency = c;
    if (currency.equals("TL") && c.equals("USD")) {
        balance = balance / 32.88;
    }
    if (currency.equals("USD") && c.equals ("TL")){
        balance = balance * 32.88;
    }
}
```

setCurrency function

```
public void setCurrency(String c) {
   if (currency.equals("TL") && c.equals("USD")) {
      balance = balance / 32.88;
   }
   if (currency.equals("USD") && c.equals ("TL")){
      balance = balance * 32.88;
   }
   currency = c;
}
```

What is the output?

```java
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, "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();
    }
}
```



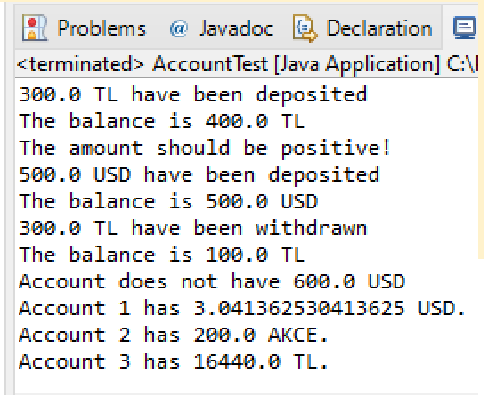
```
<terminated> AccountTest [Java Application] C:\P
300.0 TL have been deposited
The balance is 400.0 TL
The amount should be positive!
500.0 USD have been deposited
The balance is 500.0 USD
300.0 TL have been withdrawn
The balance is 100.0 TL
Account does not have 600.0 USD
Account 1 has 3.041362530413625 USD.
Account 2 has 200.0 USD.
Account 3 has 16440.0 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 / 32.88;
        }
        if (currency.equals("USD") && c.equals ("TL")){
            balance = balance * 32.88;
        }
        currency = c;
    }
```

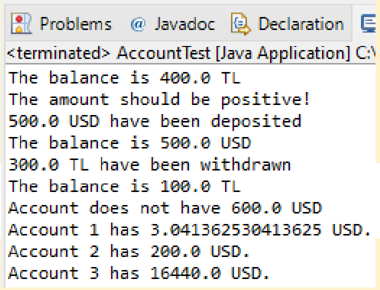


```
300.0 TL have been deposited
The balance is 400.0 TL
The amount should be positive!
500.0 USD have been deposited
The balance is 500.0 USD
300.0 TL have been withdrawn
The balance is 100.0 TL
Account does not have 600.0 USD
Account 1 has 3.041362530413625 USD.
Account 2 has 200.0 AKCE.
Account 3 has 16440.0 TL.
```

Unknown currency?

- How can we fix this setCurrency function?

```java
    public void setCurrency(String c) {
        if (currency.equals("TL") && c.equals("USD")) {
            balance = balance / 32.88;
        }
        if (currency.equals("USD") && c.equals ("TL")){
            balance = balance * 32.88;
        }
        currency = c;
    }
```



```
<terminated> AccountTest [Java Application] C:\
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 3.041362530413625 USD.
Account 2 has 200.0 USD.
Account 3 has 16440.0 USD.
```

```java
    public void setCurrency(String c) {
        if (currency.equals("TL") && c.equals("USD")) {
            balance = balance / 32.88;
        }
        if (currency.equals("USD") && c.equals ("TL")){
            balance = balance * 32.88;
        }
        if (currency.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";
        }
```

- In default we should set it to "TL"

Fixing Constructors

```
//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;
}
```

```
//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

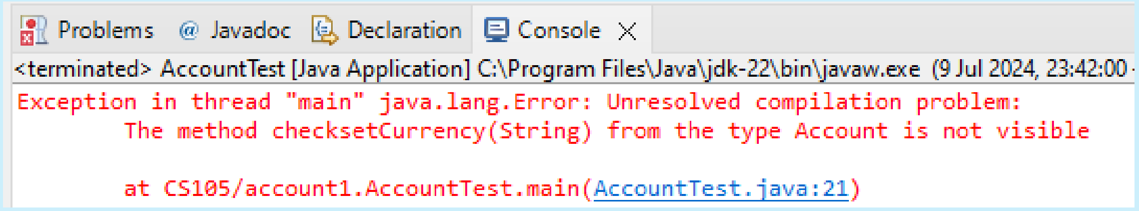
```
private void checksetCurrency(String c) {
    if (c.equals("USD"))
        currency = c;
    else
        currency = "TL";
}
```

```
//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, "USD");

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

        account2.checksetCurrency("TL");

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



```
<terminated> AccountTest [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (9 Jul 2024, 23:42:00
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
	The method checksetCurrency(String) from the type Account is not visible

	at CS105/account1.AccountTest.main(AccountTest.java:21)
```

Private Function

- Functions that 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 (Version 10)

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

```
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 (Version 10)

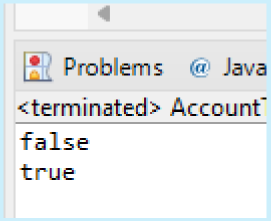
```
public int getNumber() {
     return number;
}
public double getBalance() {
     return balance;
}
public String getCurrency() {
     return currency;
}
public boolean isActive() {
     return active;
}
```

Set Functions

- 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());
}
```

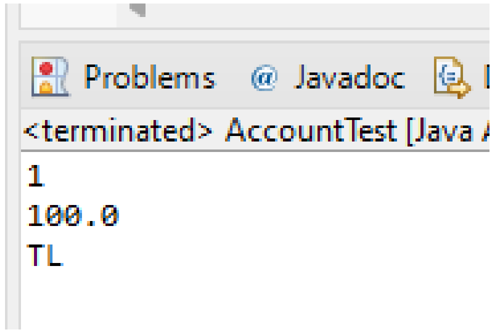


```
false
true
```

Ways of printing out the object - 1

- get methods for accessing class instances one by one

```
    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());
    }
```



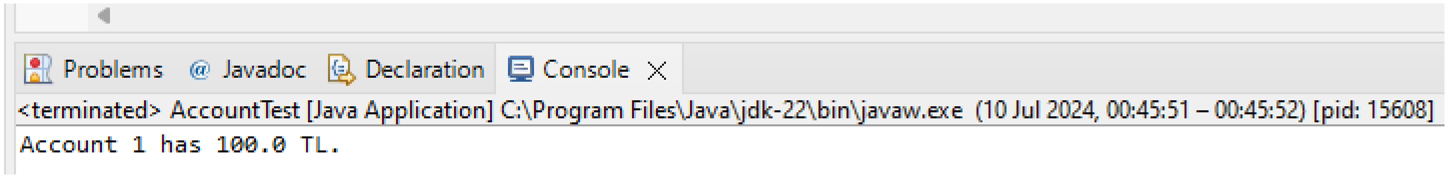
```
<terminated> AccountTest [Java
1
100.0
TL
```

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();
        }
```



Problems @ Javadoc Declaration Console

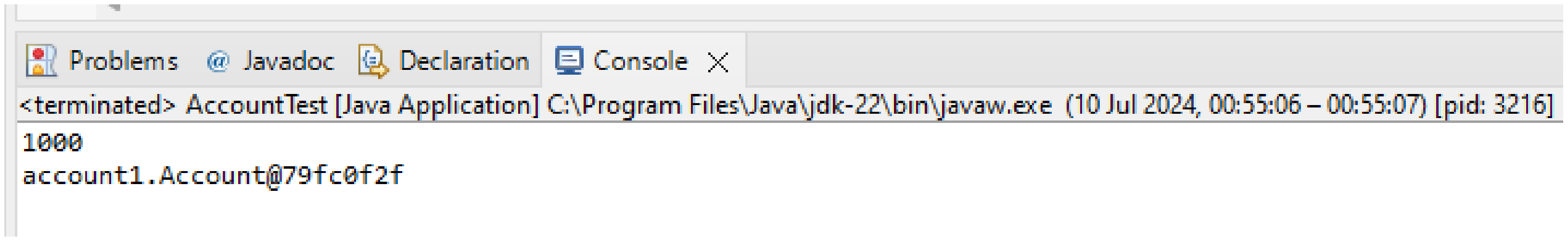
```
<terminated> AccountTest [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe  (10 Jul 2024, 00:45:51 – 00:45:52) [pid: 15608]
Account 1 has 100.0 TL.
```

Ways of printing out the object

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

```java
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?



```
<terminated> AccountTest [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (10 Jul 2024, 00:55:06 – 00:55:07) [pid: 3216]
1000
account1.Account@79fc0f2f
```

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 something meaningful, we need to override **toString** method of the class.

toString method

- toString method tells Java how to display an object of the class.

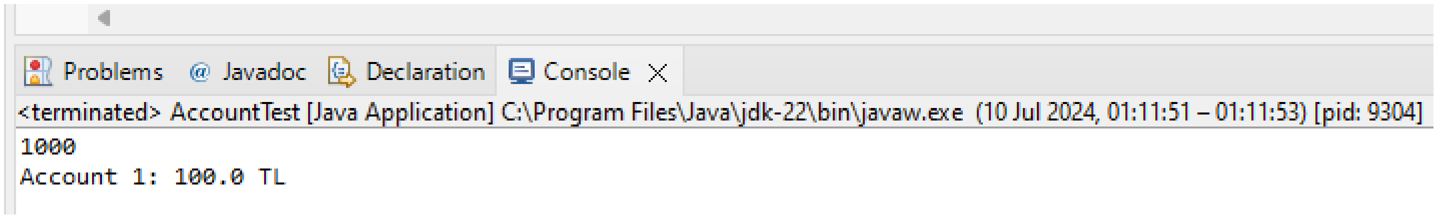
- It returns a String representation of the object.

```
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 [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe  (10 Jul 2024, 01:11:51 – 01:11:53) [pid: 9304]
1000
Account 1: 100.0 TL
```

Any Questions?