ORACLE
Academy

Java Programming

2-1
Working with Pre-Written Code

Academy

Overview

- This lesson covers the following topics:
  - Read and understand a pre-written Java program consisting of classes and interacting objects
  - Apply the concept of inheritance in the solutions of problems
  - Test classes in isolation
  - Describe when it is more appropriate to use an ArrayList than an Array

Modifying Existing Programs

- When you are programming in real-world scenarios, such as for a company, you will often maintain and modify existing programs
- In many cases, the business problems you solve will be tied to existing programs authored by other programmers
- Being able to modify an existing program is a valuable skill you will need to apply in most programming roles

If you read another person's code that is not commented, then it can make it more difficult to understand its purpose.

ORACLE
Academy

Pre-Written Code Scenario

- Imagine a banking application
- What should a banking application do?
  - Allow a user to: Create an account, deposit in the account, withdraw from the account, display the account balance, and calculate interest
- What are the components of a banking application?
  - Accounts
- What should each component do?
  - Create, deposit, and withdraw

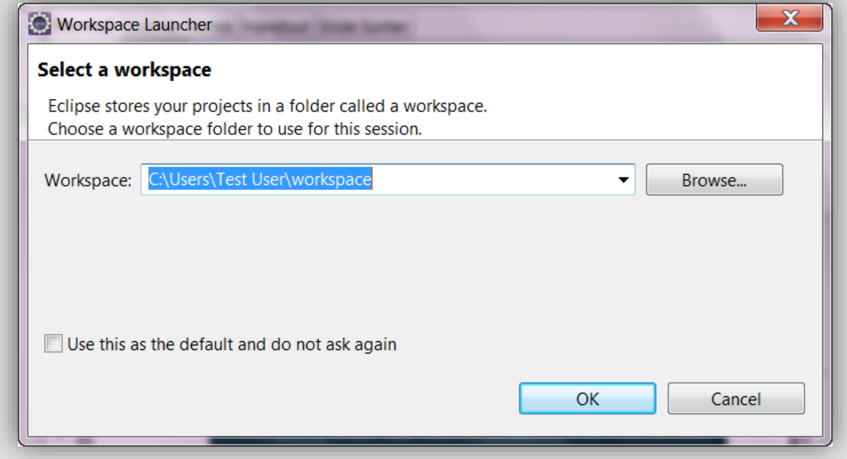
Steps to Install the JavaBank Case Study

1. Move the JavaBank.jar.zip file to a convenient location on your computer

2. Do not unzip the file!

3. Launch Eclipse

4. Select a workspace location and launch the WorkBench


Workspace Launcher

Select a workspace

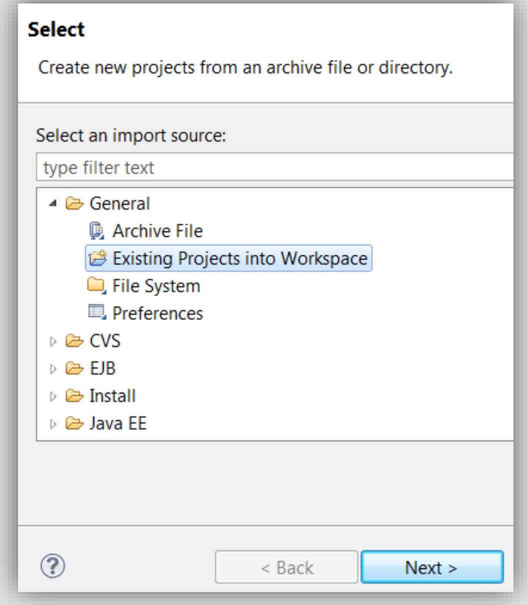
Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\Test User\workspace

ORACLE
Academy

Steps to Import Code Files into Workspace

5. Click File, then Import
6. Expand the General folder
7. Select Existing Projects into Workspace
8. Click Next
9. Click Select archive file and click Browse
10. Locate the JavaBank.jar.zip file on your local machine and Click Open



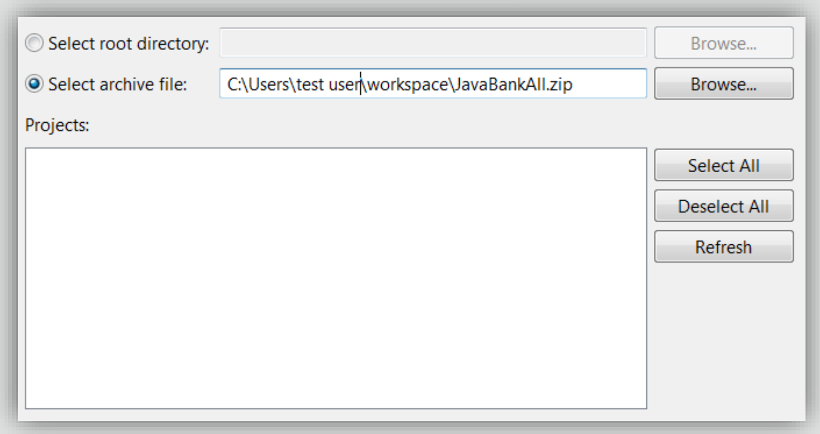
Select

Create new projects from an archive file or directory.

Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- CVS
- EJB
- Install
- Java EE



Select root directory:

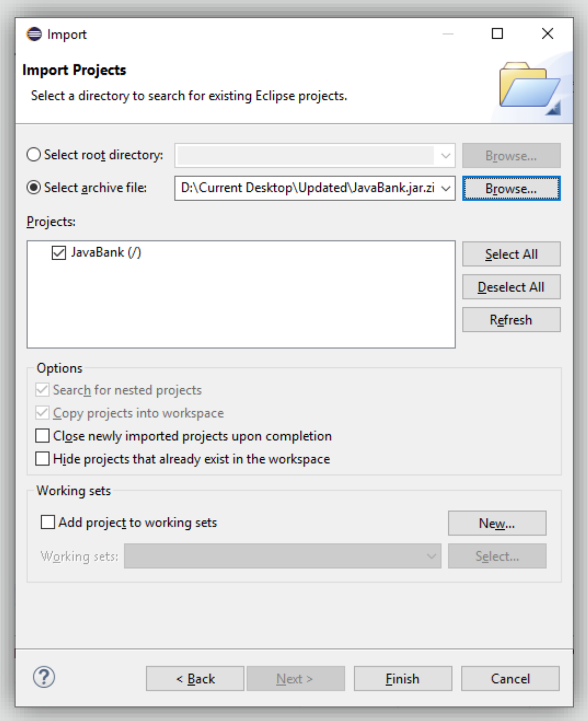
Select archive file: C:\Users\test user\workspace\JavaBankAll.zip

Projects:

ORACLE Academy

Steps to Import Code Files into Workspace

11. Ensure that the project JavaBank (/) is checked


Import

Import Projects

Select a directory to search for existing Eclipse projects.

- ( ) Select root directory: [ ] Browse...
- (●) Select archive file: D:\Current Desktop\Updated\JavaBank.jar.zi Browse...

Projects:

- [✓] JavaBank (/)

Options

- [✓] Search for nested projects
- [✓] Copy projects into workspace
- [ ] Close newly imported projects upon completion
- [ ] Hide projects that already exist in the workspace

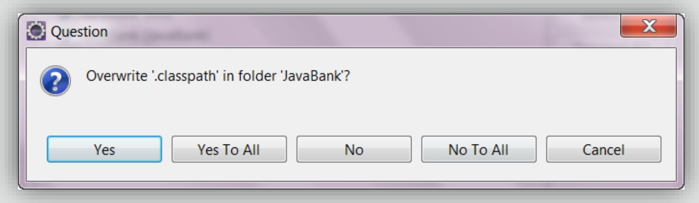
Working sets

- [ ] Add project to working sets

Working sets:

12. Click Finish

13. Click Yes to All if prompted to overwrite files

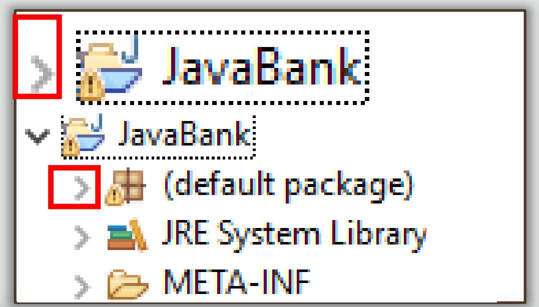

Question

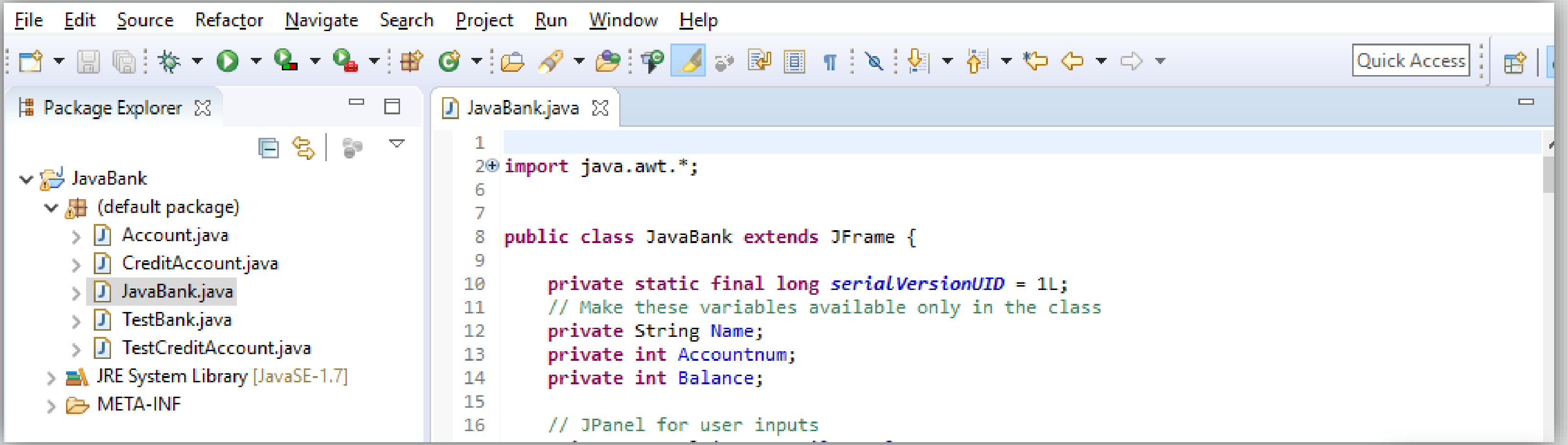
Overwrite '.classpath' in folder 'JavaBank'?

ORACLE Academy

Launching JavaBank

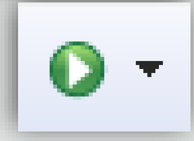
1. Expand the JavaBank project
2. Expand the default package
3. Double click the JavaBank.java file





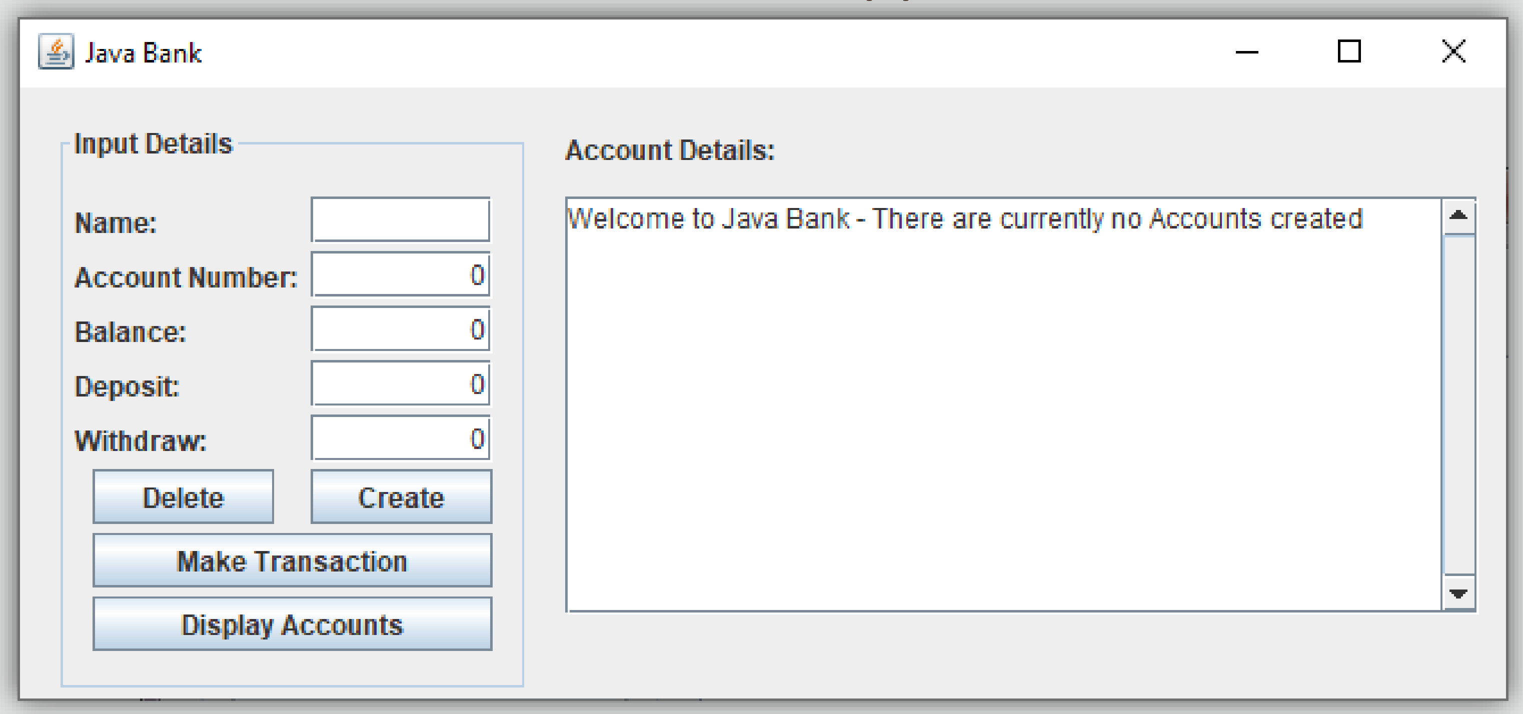
```
import java.awt.*;


public class JavaBank extends JFrame {

    private static final long serialVersionUID = 1L;
    // Make these variables available only in the class
    private String Name;
    private int Accountnum;
    private int Balance;

    // JPanel for user inputs
```

4. To launch JavaBank, compile and run the JavaBank.java application



Launching JavaBank

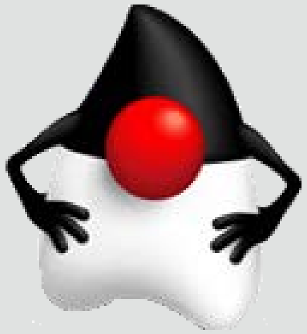
5. The JavaBank window will appear



Welcome to Java Bank - There are currently no Accounts created

- No user manual is available for this application as you will experiment with it throughout this course

Experimenting with JavaBank



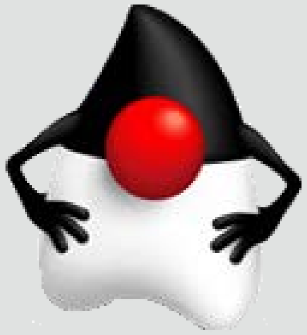
- **Quick Task**
  - Open and examine the code for the following classes:
    - Account
    - CreditAccount

- Using both classes:
  1. How many instance fields exist?
  2. How many constructors and how do they differ?
  3. What is the relationship between the classes?

Experimenting with JavaBank

- Task Solution
  1. How many instance fields exist?
     - **Account – 3**
     - **CreditAccount – 4 (1 local, 3 Super)**
  2. How many constructers and how do the differ?
     - **Account – 2, No parameters, 3 parameters.**
     - **CreditAccount - 2, No parameters, 3 parameters.**
  3. What is the relationship between the classes?
     - **Account is the superclass of the subclass CreditAccount**

Experimenting with JavaBank



- **Quick Task**
  - Look at the test driver classes:
    - testBank
    - testCreditAccount

1. Give a brief overview of the purpose of each class.
2. Can you create an Account object without having access to the CreditAccount class?
3. Can you create a CreditAccount object without having access to the Account class?

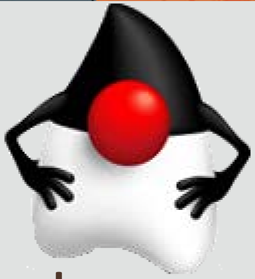
Experimenting with JavaBank

- Task Solution
  1. Give a brief overview of the purpose of each class.
     - **They test the creation and use of either accounts or credit accounts.**
     - **They also test the internal methods of each class.**
  2. Can you create an Account object without having access to the CreditAccount class? – **Yes**
  3. Can you create a CreditAccount object without having access to the Account class? - **No**

Examining Pre-Written Code

- When you receive a program (such as JavaBank) and you are unfamiliar with the code, it is important that you experiment with the application as well as examining the underlying code
- Using the application allows the identification of changes and additional functionality that can be added
- Examining the application as well as the underlying code allows you to become familiar with how the application works

Considerations When Examining Code



- When examining code, keep the following in mind:
- Look for comments in the code

```
// Single line comments are preceded by a double forward slash

/* Comments spanning more than one line are enclosed by
backslashes and asterisks */
```

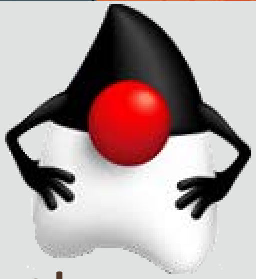
- Task: Identify the classes
  1. Examine each class and read the comments to gain an understanding of the program structure

A class is a blueprint for an object.
A class describes what an object knows and what an object does.

Techniques When Examining Code

- Here are some other techniques to try when reading code:
  - Re-run the application
  - Learn the high-level structure of the code and then find the point of entry and how it branches from there
  - Understand the constructs
  - Perform some testing
  - If you still have trouble understanding the code, ask someone else for their thoughts
  - Reach out to other programmers in a programmer's forum

The Account Class



- **Quick Task**: Examine the Account class and note the number of:

1. Constructors
   - What values are assigned to the fields in each constructor?
2. Methods
   - How many methods are accessors?
   - How many methods are mutators?

An accessor is a method that can access the contents of an object but does not modify that object.
A mutator is a method that can modify an object.

Constructors in Account Class

- Task Solution: 1. Constructors
  - The default constructor sets:
    - Account Name to NULL (Empty)
    - Account Number to 0
    - Balance to 0.

```
Account()
```

  - The overloaded constructor takes values as parameters and sets these values to the instance of Account being created

```
Account(String name, int num, int amt)
```

Methods in Account Class

Task Solution: 2. Methods

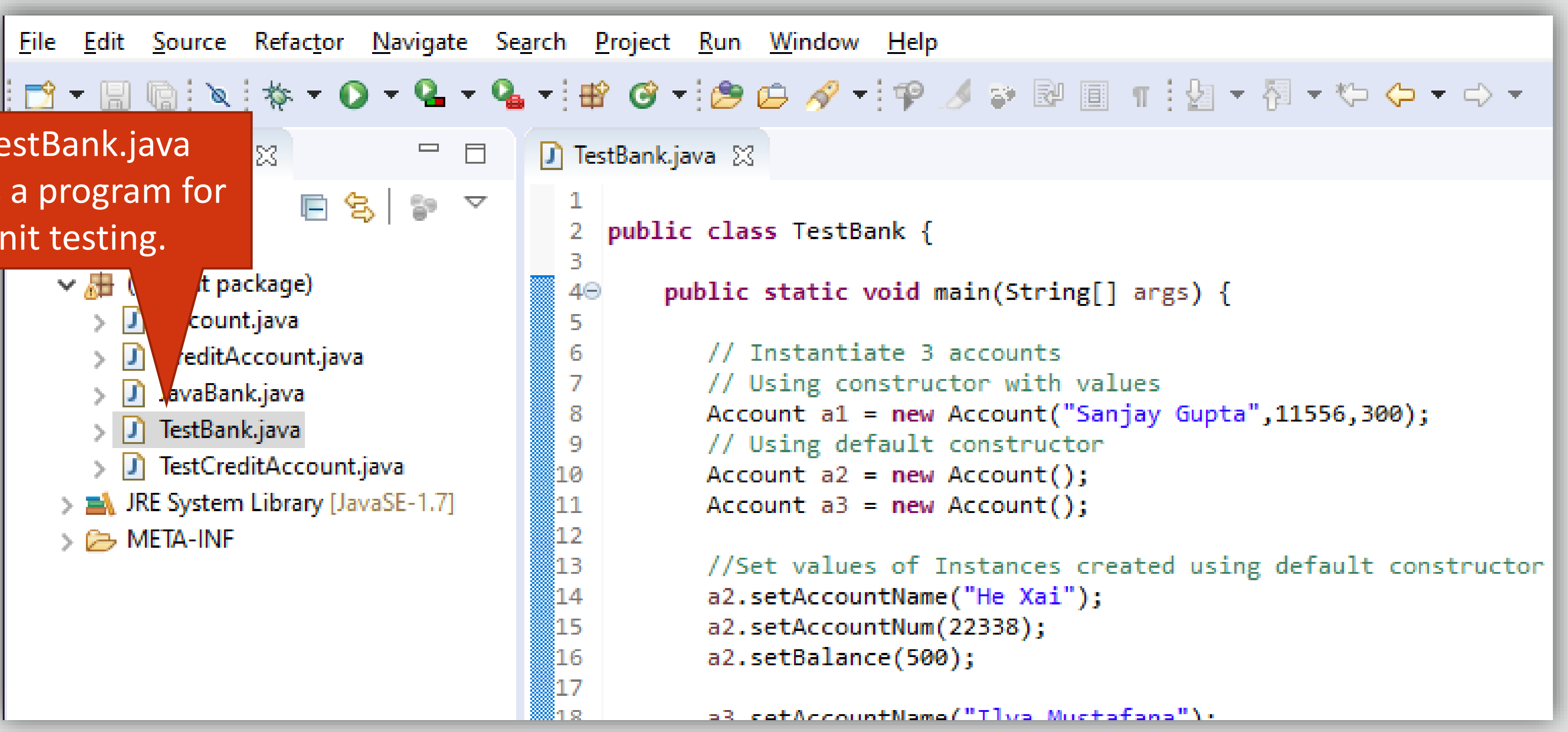
Method	Description
`public void deposit(int amt)`	Updates the balance with a deposit amount
`public void withdraw(int amt)`	Updates the balance with a withdrawal amount
`public void setAccountName(String name)`	Sets the account name value
`public void setAccountNum(int num)`	Sets the account number value
`public void setBalance(int num)`	Sets the account balance value
`public String getAccountName()`	Returns the account name value
`public int getAccountNum()`	Returns the account number value
`public int getBalance()`	Returns the account balance value
`public void print()`	Prints the instance field values, this is included to accommodate isolation testing

Testing Classes in Isolation

- When you create a class, it is good practice to test the class independently before testing it within the application to detect problems in that code
  - This is known as isolation testing or unit testing
  - The main purpose of unit testing is to verify that an individual unit (a class, in Java) is working correctly before it is combined with other components in the system
- After creating the class, test it in isolation by creating a small program that calls the constructors, modifiers, and accessors

Unit Test Program Example

- The example unit test program (testBank.java) below has a main and creates instances of the Account class


File Edit Source Refactor Navigate Search Project Run Window Help
TestBank.java is a program for unit testing.
t package)
count.java
reditAccount.java
JavaBank.java
TestBank.java
TestCreditAccount.java
JRE System Library [JavaSE-1.7]
META-INF
TestBank.java

```java

public class TestBank {

    public static void main(String[] args) {

        // Instantiate 3 accounts
        // Using constructor with values
        Account a1 = new Account("Sanjay Gupta",11556,300);
        // Using default constructor
        Account a2 = new Account();
        Account a3 = new Account();

        //Set values of Instances created using default constructor
        a2.setAccountName("He Xai");
        a2.setAccountNum(22338);
        a2.setBalance(500);

        a3.setAccountName("Ilya Mustafana");
```

Oracle Academy

Testing Classes in Isolation Example

```
public class TestBank {
   public static void main(String[] args) {
      // Using constructor with values
      Account a1 = new Account("Sanjay Gupta",11556,300);
      Account a2 = new Account(); // Using default constructor
      Account a3 = new Account(); // Using default constructor
      //Set values of Instances created using default constructor
      a2.setAccountName("He Xai");
      a2.setAccountNum(22338);
      a2.setBalance(500);

      a3.setAccountName("Ilya Mustafana");
      a3.setAccountNum(44559);
      a3.setBalance(1000);

      // Print accounts
      a1.print();
      a2.print();
      a3.print();
   }//end method main
}//end class testBank
```

The Deposit Method

- This is the code for the deposit method from the Account class

```
public void deposit(int amt)
{
   balance = balance + amt;
}//end method deposit
```

- When this method is called, the value from the edit box is passed in as amt and is added to the balance field of the current account instance
- Similar actions are performed by the withdraw, setAccountName, setAccountNum and setBalance methods

The getaccountname Method

- Below is the code for the getAccountName method

```java
public String getAccountName()
{
   return accountName;
}//end method getAccountName
```

- When getAccountName() is called, the value of accountName for the current account instance is returned to the calling method
- Similar actions are performed by the getAccountNum and getBalance methods

Inheritance

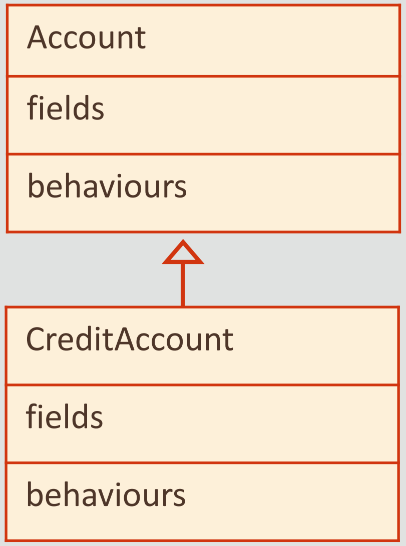
- Inheritance is when you have one class that is a parent class (called a superclass) and another class that is a child class (called a subclass)
- The child class is said to be derived from the parent class
- The reason to have a child class is to keep information separate
- The child can inherit all the methods and fields from its parent, but can then act independently

Inheritance can be defined as the process where one object acquires the properties of another.

ORACLE Academy

Extending the Account Class

- Let's assume that you want to create a new account type that behaves differently from a standard account
- To create this type, you can extend the Account class
- Consider a credit account
- A subclass that handles information about the credit amount associated with an account can be created as a subclass

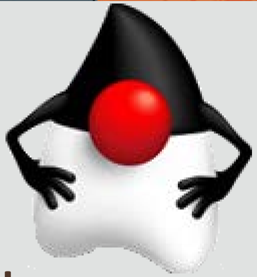


Using the extends Keyword

- Use the keyword extends when creating a subclass that extends an existing class

```java
public class CreditAccount extends Account{

}//end class CreditAccount
```

- This will extend the Account class as a Credit Account
- It will have the same behavior as a standard account but will add the ability to set the credit limit
- The CreditAccount class will inherit all of the methods from Account class



Inheritance Task

- **Quick Task:** Examine the relationship between the Account class and the CreditAccount class
  1. Instance Fields
     - How many instance fields are associated with a credit account?
  2. Constructors
     - What order are the classes constructed in?
  3. Methods
     - How is it possible that the CreditAccount print method can access the instance fields in the Account class?

Inheritance Suggested Solution

1. Instance Fields
   - How many instance fields are associated with a credit account? - **4 instance fields (1 in CreditAccount, 3 in Account)**
2. Constructors
   - What order are the classes constructed in? – **Account – CreditAccount**
3. Methods
   - How is it possible that the CreditAccount print method can access the instance fields in the Account class? – **Fields in the Account class have default visibility**

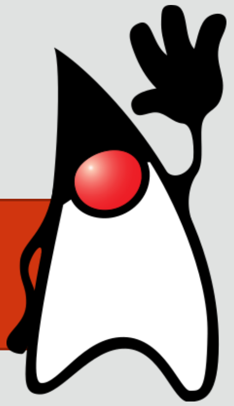
ArrayList or Array?

- In Java, an Array has a fixed length
- Once the array is created, the programmer needs to know the length of the Array because it cannot grow or shrink in size
- If you have a situation where you cannot predict the number of objects that you will be storing, you could use an ArrayList instead of using a fixed length Array

ArrayList Operations

- In an Array, you need to know the length and the current number of elements stored
- In an ArrayList you can use predefined behaviors to perform these operations
  - **IsEmpty** : Returns true if this list contains no elements
  - **size** : Returns the number of elements in this list

An ArrayList is an Array that can store multiple object types and can grow and shrink dynamically as required.



Other ArrayList Operations

ArrayList Operation	Description
add	Appends to the end of this list.
clear	Removes all of the elements from this list.
contains	Returns true if this list contains the specified element.
get	Returns the element at the specified position in this list.
remove	Removes the element in this list.
set	Replaces the element at the specified position in this list.
trimToSize	Trims the capacity of this ArrayList instance to be the list's current size.

ArrayList or Array?

- The JavaBank application uses Arrays to store the data in the accounts
- The create, withdraw, deposit, and display methods manipulate the data to produce the desired result

This makes it difficult to continually add new accounts as you will run out of space in your fixed length array.

ArrayList in JavaBank

- In the JavaBank application, you can use an ArrayList in place of the myAccounts Array to:
  - Dynamically store accounts
  - Store both savings and credit accounts
- Use the ArrayList operations to add, delete, search, and so on

Using an ArrayList reduces the amount of code required to work with your data.

Terminology

- Key terms used in this lesson included:
  - Accessors
  - Arraylist
  - Inheritance
  - Isolation testing
  - Mutators

Summary

- In this lesson, you should have learned how to:
  - Read and understand a pre-written Java program consisting of classes and interacting objects
  - Apply the concept of inheritance in the solutions of problems
  - Test classes in isolation
  - Describe when it is more appropriate to use an ArrayList than an Array

ORACLE
Academy