



# Java Programming

5-4

Working with Pre-Written Code





# 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.

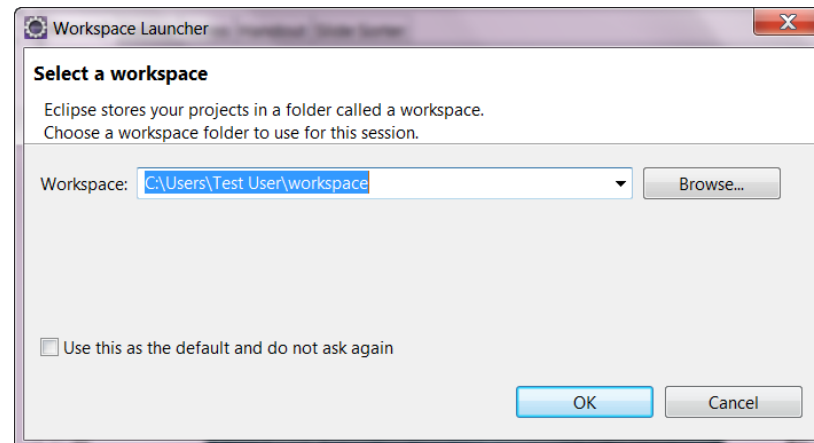


# Pre-Written Code Scenario

- 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

- Move the JavaBank.jar.zip file to a convenient location on your computer.
- Do not unzip the file.
- Launch Eclipse.
- Select a workspace location and launch the WorkBench.

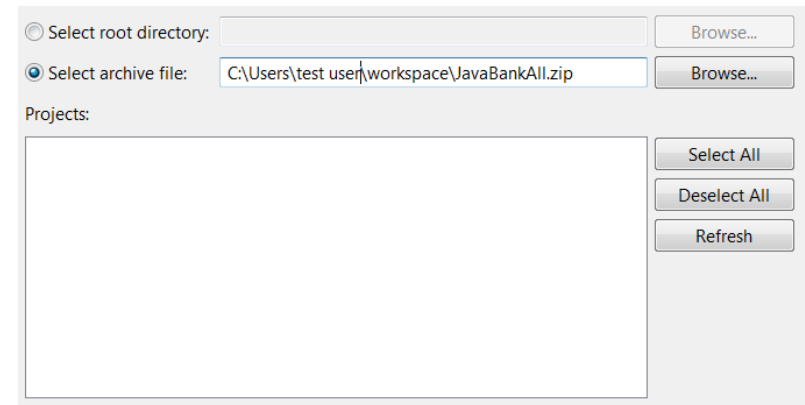
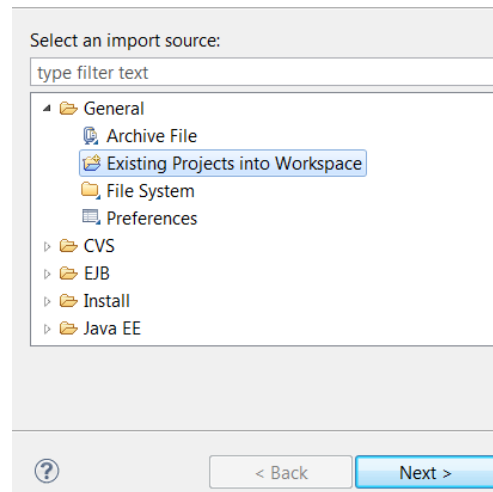


# Steps to Import Code Files into Workspace

- Click File, then Import.
- Expand the General folder.
- Select Existing Projects into Workspace.
- Click Next.

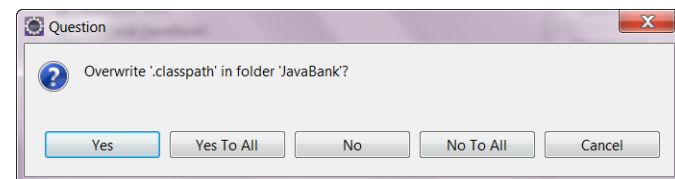
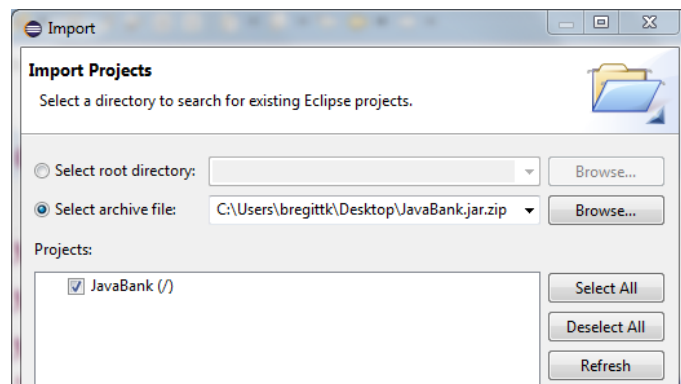
## Select

Create new projects from an archive file or directory.



# Steps to Import Code Files into Workspace

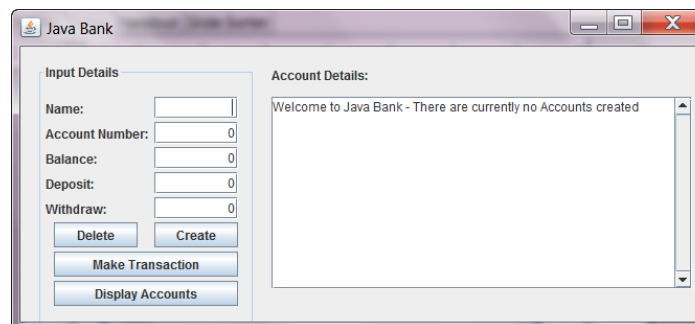
- Click Select archive file and click Browse and locate the JavaBank.jar.zip file. Click Open.
- Ensure that the project JavaBank (/) is checked.
- Click Finish.
- Click Yes to All when prompted to overwrite files.





# Launching JavaBank

- To launch JavaBank, compile and run the JavaBank.java application.
- The JavaBank window will appear.
- When you launch a program written by someone else, there is usually a user manual.
- No user manual is available because it is up to you to experiment with the application.





# Experimenting with JavaBank

- Run the program.
- Look at the classes.
  - Account
  - CreditAccount
- Look at the test driver classes (testBank and testCreditAccount) and the JavaBank application.
- This will aid your understanding as you begin to investigate the underlying code.



# 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 before examining the underlying code.
- You have already tried using the application and listed changes and additional functionality you would like to add.
- Now examine the underlying code 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 back slash  
  
/* Comments spanning more than one line are enclosed by  
backslashes and asterisks */
```

- Identify the classes.
- Examine each class and read the associated 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

- Examine the Account class and note the number of:
- Constructors
- 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

The Account class has two constructors:

- The default constructor `Account()` sets the Account Name to “EMPTY”, the Account Number to 0 and the Balance to 0.
- The overloaded constructor `Account(String name, int num, int amt)` takes values as parameters and set these values to the instance of Account being created.

# Methods in Account Class

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 account name, account number and account balance value. This is included to accommodate isolation testing.



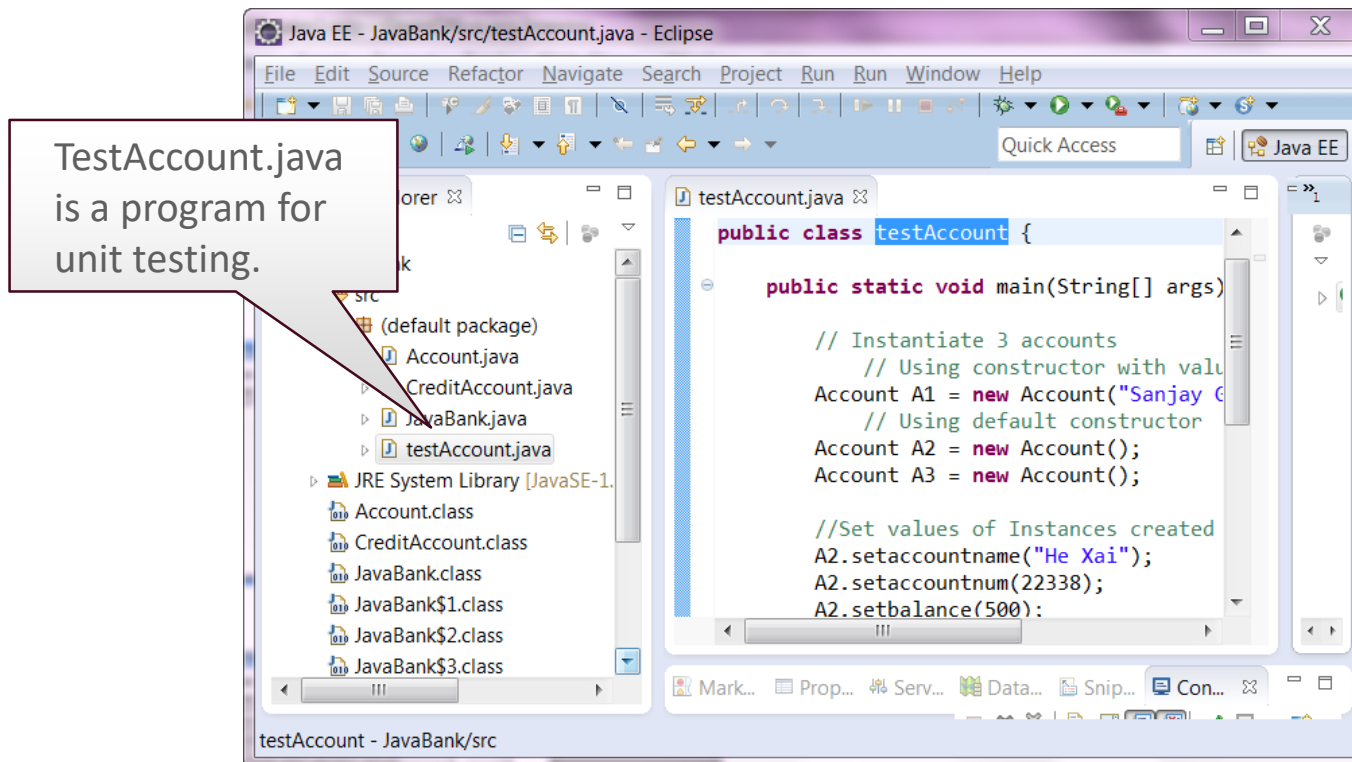


# Testing Classes in Isolation

- 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 below has a main and creates instances of the Account class.



# Testing Classes in Isolation Example

```
public class testAccount {  
    public static void main(String[] args) {  
        // Instantiate account using constructor with values  
        Account A1 = new Account("Sanjay Gupta",11556,300);  
        // Instantiate account using default constructor  
        Account A2 = new Account();  
        Account A3 = new Account();  
        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();  
    }  
}
```

# The Deposit Method

- Below is the code for the deposit method.

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

- When this method is called, the value from the edit box is passed in as amt and is added to the balance of the current account instance.
- Similar actions are performed by the withdraw and the setaccountname, setaccountnum and setbalance methods.

# The getaccountname Method

- Below is the code for the getaccountname method.

```
public String getaccountname ( )  
{  
  
    return accountname;  
}
```

- When this method is called, the value of accountname of 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.

# 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.
- We will create a subclass that handles information about the credit amount associated with an account.

# Using the extends Keyword

- Use the keyword extends when creating a new subclass to extend an existing class.
- This will extend the Account class as a Credit Account that 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.

```
public class CreditAccount extends Account{  
}
```





# 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.

# 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, then instead of using an Array, you can use an ArrayList.

# 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 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.
- Reduce the amount of code required.

# 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

