Федеральное государственное автономное образовательное учреждение

высшего образования

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа «Киберфизические системы и управление»

**Отчет №2**

по дисциплине «Системный подход к разработке программного обеспечения»

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. 3530902/00201 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Перевалов В. В. |
|  | <*подпись*> |  |
|  |  |  |
|  |  |  |
| Руководитель:  Кандидат т.н. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Нестеров С. А. |
|  | <*подпись*> |  |

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2023 г.

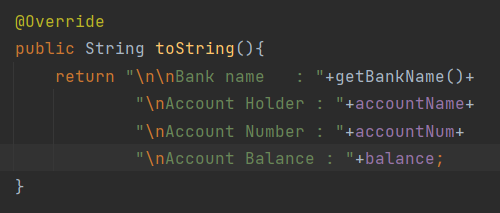
Санкт-Петербург

2023

**Ход работы**

**Слайды урока 2\_3**

Слайд 40



Слайд 41

Изображение выглядит как текст

Автоматически созданное описание

Переопределили метод toString для класса CreditAccount.

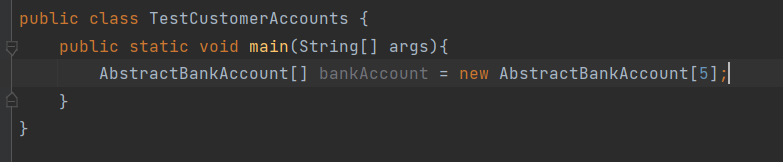
Слайд 42

Изображение выглядит как текст

Автоматически созданное описание

Теперь можем использовать такую запись вместо вызова метода print.

Слайд 45



Создали массив данных типа AbstractBankAccount.

Слайд 46

Изображение выглядит как текст

Автоматически созданное описание

Слайд 47

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Слайд 48

Изображение выглядит как текст

Автоматически созданное описание

С помощью оператора instanceof мы можем «распечатать» только аккаунты, минуя кредитные аккаунты.

Слайд 51

Изображение выглядит как текст

Автоматически созданное описание

То же самое для кредитных аккаунтов.

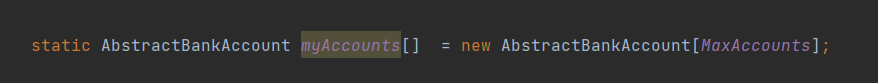
**2-3: Java Class Design – Abstract Classes**

**Vocabulary**

|  |  |
| --- | --- |
| Downward cast | The type of casting that changes a generalized object to a more specialized object type |
| Virtual Method Invocation | The process of a call to a generalized method and actually calls the instantiated subclass method, or appropriate subclass method. |
| instanceof | The operator that allows you to compare a class instance against a class type |
| Casting | The process of explicitly changing one data type to another data type |
| Abstract class | A class with an abstract constructor and at least one method that is defined but not implemented. |
| Upward cast | This type of casting changes a specialized object instance into a generalized instance. It doesn't lose any of its detail but you can't access them without downcasting the object to access specialized methods |
| Abstract class constructor | A constructor without implementation that makes the class restricted in that it cannot create instances. |

1. Update the JavaBank.java application to use the toString() methods to display the bank account details to the text area in the Java application.

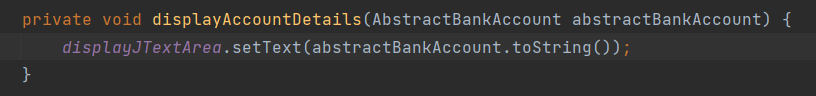
a) Update the myAccounts array definition to use the AbstractBankAccount class as its base class.



b) Update the displayAccountDetails() method to accept a single parameter of type AbstractBankAccount named account

c) Call the account objects toString() method to provide the text for the JTextArea.

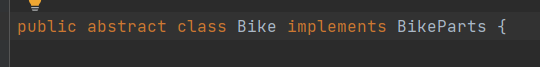
d) Update the method calls to displayAccountDetails() to pass a single account object as an argument.



2. Give one reason why you might use an Abstract class rather than an Interface

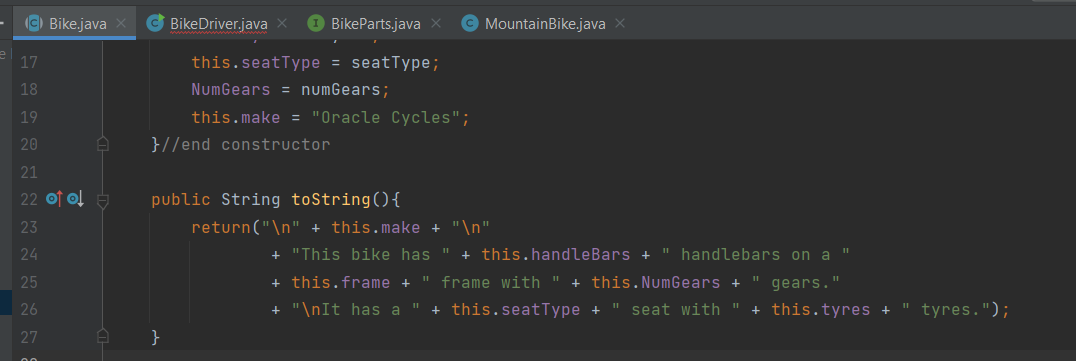
Абстрактный класс может реализовывать методы, которые могут быть использованы или переопределены классами-наследниками. Это бывает полезно. Интерфейсы позволяют определить функциональность, но не использовать.

3. Currently in your bikeproject you can instantiate an object based on the super class Bike. Update the Bike class so that you cannot create a Bike object.



4. Remove the bike4 code from the driver class.

5. Convert the printDescription methods in the classes to toString methods and update the code that displays the object values to the console.



Изображение выглядит как текст

Автоматически созданное описание

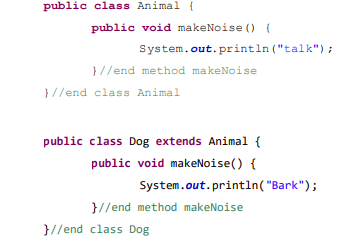
Изображение выглядит как текст

Автоматически созданное описание

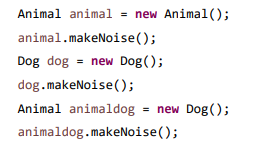
Изображение выглядит как текст

Автоматически созданное описание

6. Given the following classes.



a) What would the output of the following be? Explain your answer.

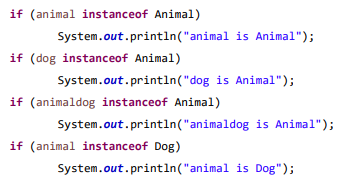


“talk”-от экземпляра суперкласса

“bark”-от подкласса, метод makeNose переопределен

“bark”- animaldog – переменная, являющаяся экземпляром класса Dog

b) Using the animal and dog classes above. If we added the following code to the driver what would the output be:



выполнятся все условия, кроме последнего, потому что во всех них объект -потомок суперкласса Animal.

7. Describe casting both for primitives and objects

Приведение типов это изменение типа переменной. При приведении примитивных типов данных переменная либо увеличивает количество доступных ей значений, либо уменьшает, при этом округляется ее значение до значения нового типа, при понижении могут случаться ошибки.

Приведение объектов это изменение класса либо поднятие класса переменной до суперкласса, при этом теряется доступ к методам подкласса, либо изменение класса с супер до подкласса.

8. downcasting:

Animal animal=new Animal();

Dog dog1=(Dog)animal;

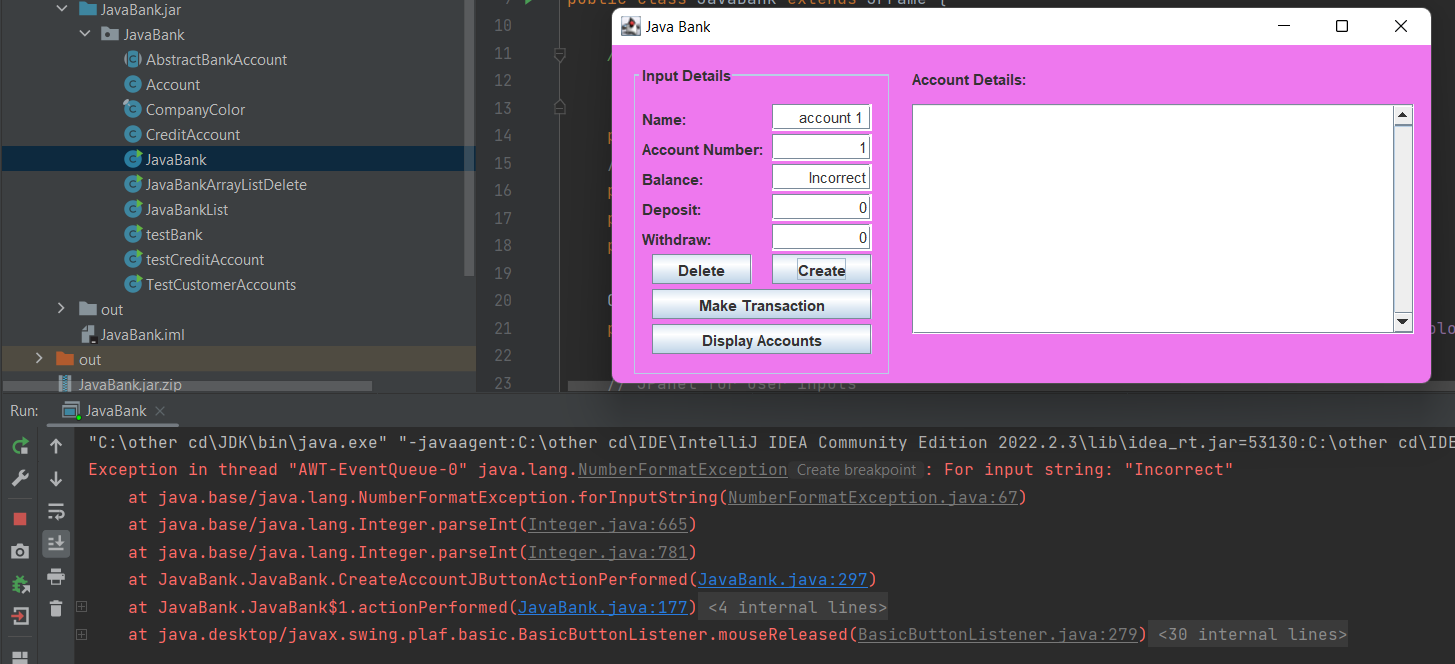
upcasting:

Dog dog=new Dog();

Animal animal=(Animal)doge;

**2-4 Exceptions and Assertions**

Слайд 6



Слайд 8

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, внутренний, ноутбук

Автоматически созданное описание

Исправили ситуацию с помощью блока try-catch.

Слайд 13

Изображение выглядит как текст

Автоматически созданное описание

Применение множественных блоков catch. Такое применяется, если каждое исключение нужно обработать собственным образом.

Слайд 16

Изображение выглядит как текст

Автоматически созданное описание

Добавление блока finally, который выполнится независимо от того, было ли поймано исключение или нет.

Cлайд 19

Изображение выглядит как текст

Автоматически созданное описание

С помощью такой конструкции в одном блоке catch могут быть пойманы 2 вида исключений.

**2-4: Exceptions and Assertions**

**Practice Activities**

|  |  |
| --- | --- |
| try-with-resources | A certain kind of try statement that contains resources. |
| invariant class | An invariant used to evaluate the assumptions of the class instances. |
| Conditional statements | Certain types of boolean statements that allow you to test specific aspects of your code |
| try and catch block | Key statement for handling exceptions in Java. |
| Internal Invariants | An invariant that handles boolean statements to test internal values. |
| Control Flow Invariants | An invariant that handles conditions in control flow statements. |
| Multi-Catch Statement | A statement that allows you to handle multiple exceptions. |
| Finally Clause | An optional addition to a try-catch statement that will always be executed. |
| Method-declared exceptions (throws) | Run-time errors that can be handled inside the program. |

1. You have included exception handling for the create button in the JavaBank application. Do the same for the make transaction button.

Изображение выглядит как текст

Автоматически созданное описание

2. Create an exception class called “myException” that accepts a String message as a parameter in its constructor

and passes the message to the super class to be printed out when an error message is thrown.

Изображение выглядит как текст

Автоматически созданное описание

3. Update all of the catch(Exception e)statements to create a MyException object named newExc that sends the message "An unhandled error occurred!!" to the console .

Изображение выглядит как текст

Автоматически созданное описание

4. Create a block of code that utilizes all three types of invariants and asserts their values.

package asserts;  
  
  
import java.util.Scanner;  
  
public class AssertsTester {  
 public static void main(String[] args){  
 //First();  
 //Second();  
 *Third*();  
 }  
  
 public static void First(){  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Введите неотрицательное число: ");  
  
 int value = scanner.nextInt();  
 scanner.close();  
  
 assert (value>=0):"Надо было неотрицательное.....";  
 System.*out*.println("Отлично!");  
 }  
  
 public static void Second(){  
 Scanner scanner2 = new Scanner(System.*in*);  
 System.*out*.print("Выберете 7 или 13 :");  
 int value = scanner2.nextInt();  
  
 scanner2.close();  
  
 switch (value){  
 case 7:  
 System.*out*.println("Счастливое!");  
 break;  
 case 13:  
 System.*out*.println("Не повезло!");  
 break;  
 default:  
 assert (false);  
 break;  
 }  
 }  
  
 public static void Third(){  
 System.*out*.print("Введите \"Привет\" :");  
  
 Scanner scanner = new Scanner(System.*in*);  
  
 String line = scanner.nextLine();  
 scanner.close();  
  
 assert (line.equals("\"Привет\"")): "Неверно :(";  
 System.*out*.println("Все верно!");  
 }  
}

Чтобы ассерты срабатывали, следует запускать из консоли с опциями -ea.