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 по лабораторной работе**

по дисциплине «Системный подход к разработке программного обеспечения»

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

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2023 г.

Санкт-Петербург

2023

**Примеры из книги Васильева**

**Глава 8**

**Пример 1**

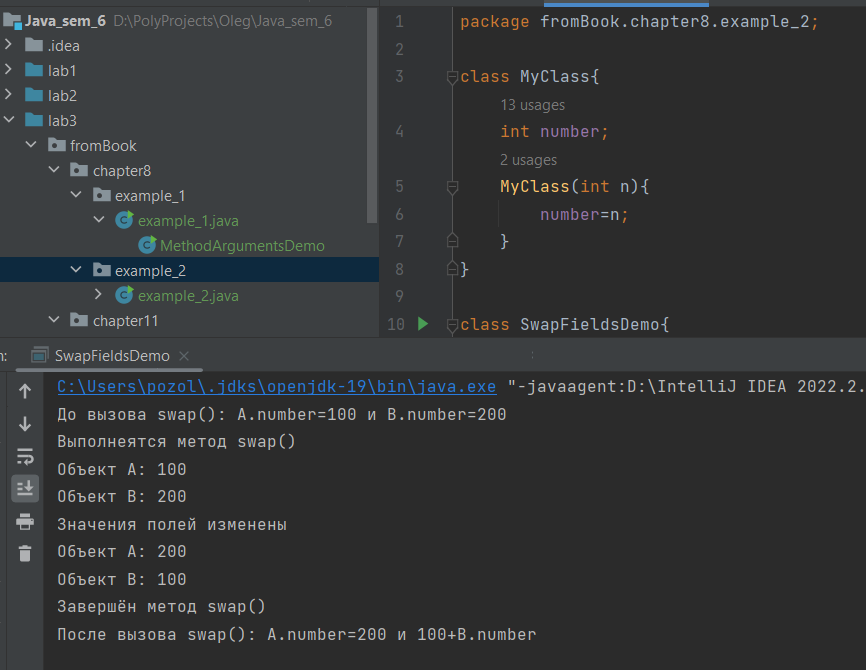
Базовое положение состоит в том, что при передаче аргументов методам на самом деле передается техническая, автоматически создаваемая копия аргументов . Чтобы пояснить и раскрыть суть проблемы, рассмотрим небольшой пример в листинге 8.1.

**Изображение выглядит как текст

Автоматически созданное описание**

**Пример 2**

Теперь рассмотрим фактически тот же пример, что и выше, но только метод swap() будет описан так, что аргументами ему передаются не целые числа, а объекты (если быть более точным, то объектные переменные, или ссылки на объекты). Рассмотрим программный код в листинге 8.2.

****

**Пример 3**

В представленном далее программном коде, по сравнению с примером из листинга 8.2, принципиально изменился лишь код метода swap(): если ранее мы изменяли значения полей объектов, то теперь пытаемся изменить ссылки на объекты. Рассмотрим код в листинге 8.3 (поскольку он во многом похож на код из листинга 8.2, то для сокращения объема кода основная часть комментариев удалена, а наиболее важное место кода выделено жирным шрифтом).

**Изображение выглядит как текст

Автоматически созданное описание**

**Пример 4**

Нередко результатом метода должен возвращаться объект. Схема, по которой результатом метода возвращается объект, сводится к тому, что при выполнении метода создается объект, а результатом метода возвращается ссылка на этот объект, которая «технически» реализуется через объектную переменную. Простой пример описания метода, возвращающего результатом объект, представлен в листинге 8.4.

**Изображение выглядит как текст

Автоматически созданное описание**

**Пример 5**

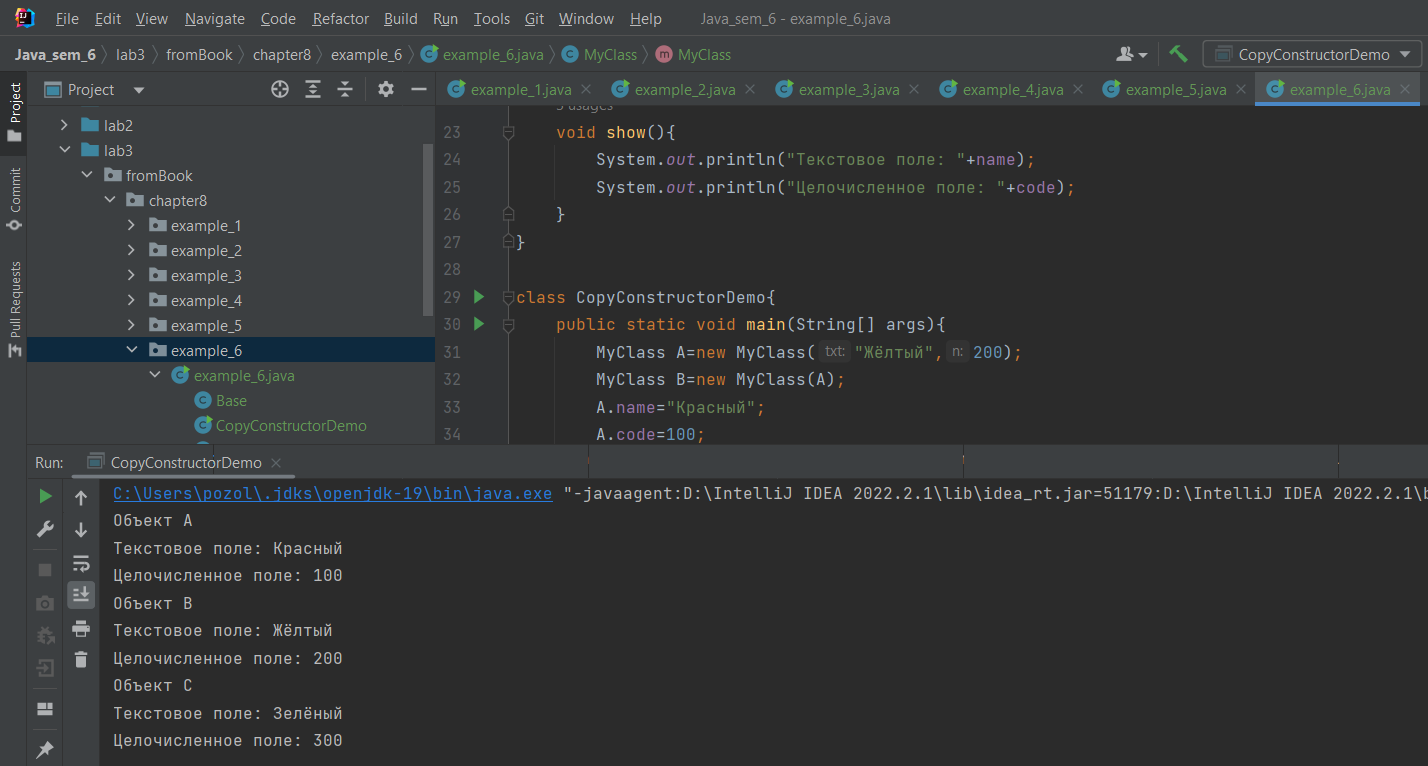
Напомним, что объектная переменная суперкласса может ссылаться на объект подкласса. Мы воспользуемся этим замечательным обстоятельством для того, чтобы создать метод, возвращающий, в зависимости от фактического значения своего аргумента, объекты разных классов. Рассмотрим программный код, представленный в листинге 8.5.

**Изображение выглядит как текст

Автоматически созданное описание**

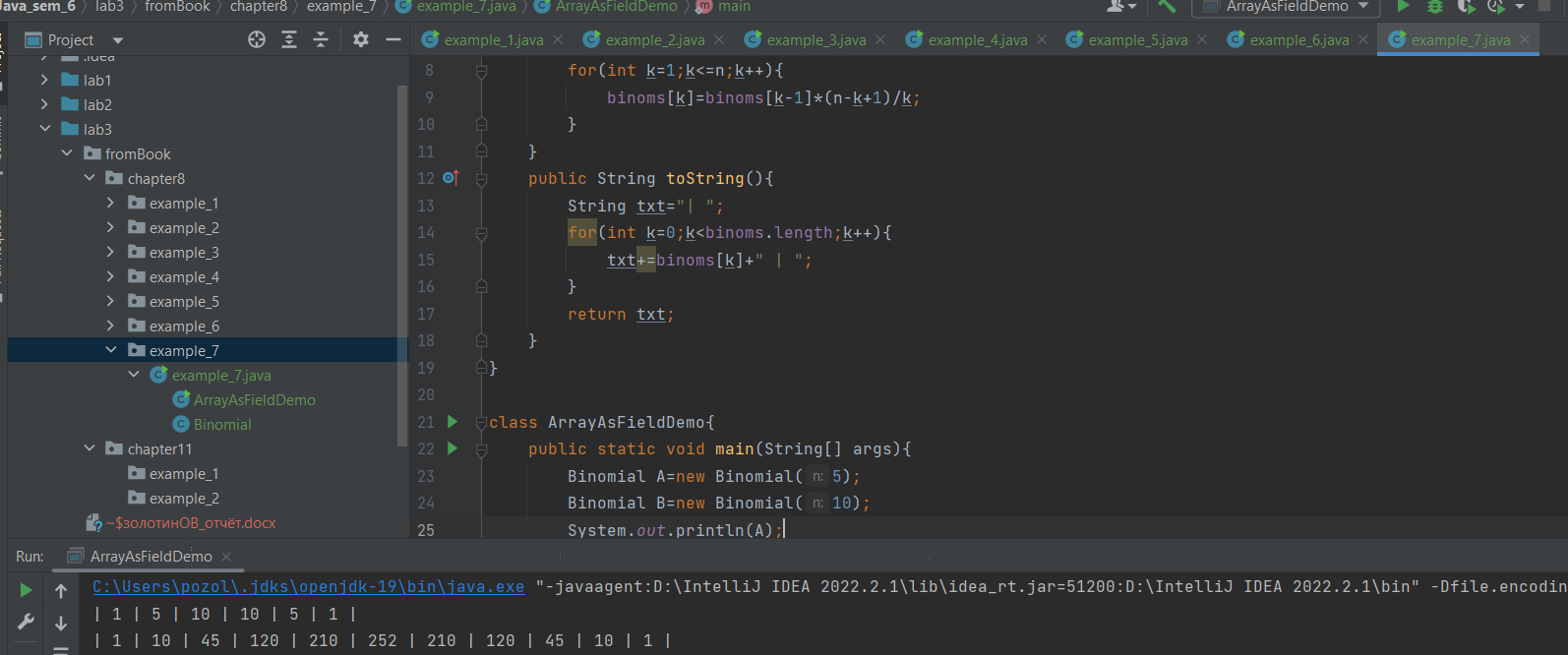
**Пример 6**

Еще один, важный с практической точки зрения, пример иллюстрирует использования в классе конструктора создания копии . Речь идет об описанном в классе конструкторе, который позволяет создавать на основе уже существующего объекта другой объект (обычно с такими же значениями полей, но это не обязательно). В принципе, ситуация достаточно тривиальная, но если при этом используется наследование, то могут возникнуть некоторые не очень очевидные «моменты». Чтобы не быть голословными, сразу перейдем к рассмотрению примера. Интересующий нас программный код представлен в листинге 8.6.



**Пример 7**

Итак, допустим, что необходимо описать класс, у которого полем является массив. Как это можно сделать? В принципе, достаточно просто: в классе полем объявляется переменная массива, а создание и заполнение собственно массива обычно переносится в конструктор. Небольшой пример, поясняющий, как такое происходит, представлен в листинге 8.7

****

**Пример 8**

При создании массива объектов массив создается из объектных переменных, а уже затем в каждую из переменных записывается ссылка на объект. Далее мы рассмотрим небольшой пример, связанный, как и в предыдущем случае, с вычислением биномиальных коэффициентов. Только если в примере из листинга 8.7 описывался класс с полеммассивом, то теперь будет создаваться массив из объектов, у каждого из которых имеется числовое поле, значение которого вычисляется как значение биномиального коэффициента. Рассмотрим программный код в листинге 8.8.

**Изображение выглядит как текст

Автоматически созданное описание**

**Пример 9**

Еще один способ организации объектов, не подразумевающий применения массива, состоит в том, что создается «цепочка» объектов: это группа объектов, в которой каждый объект (за исключением последнего) содержит ссылку на другой объект. Применение такого подхода проиллюстрируем на примере. Рассмотрим программный код из листинга 8.9.

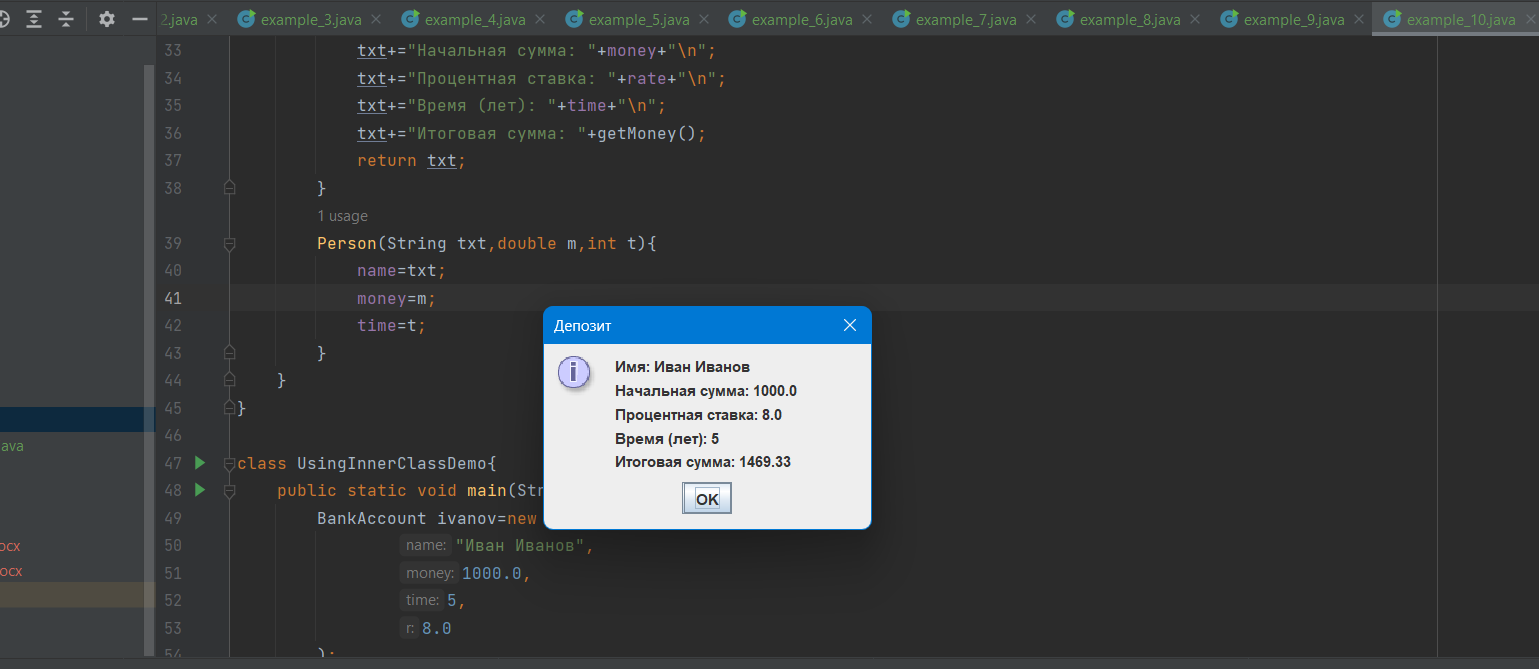
**Изображение выглядит как текст

Автоматически созданное описание**

**Пример 10**

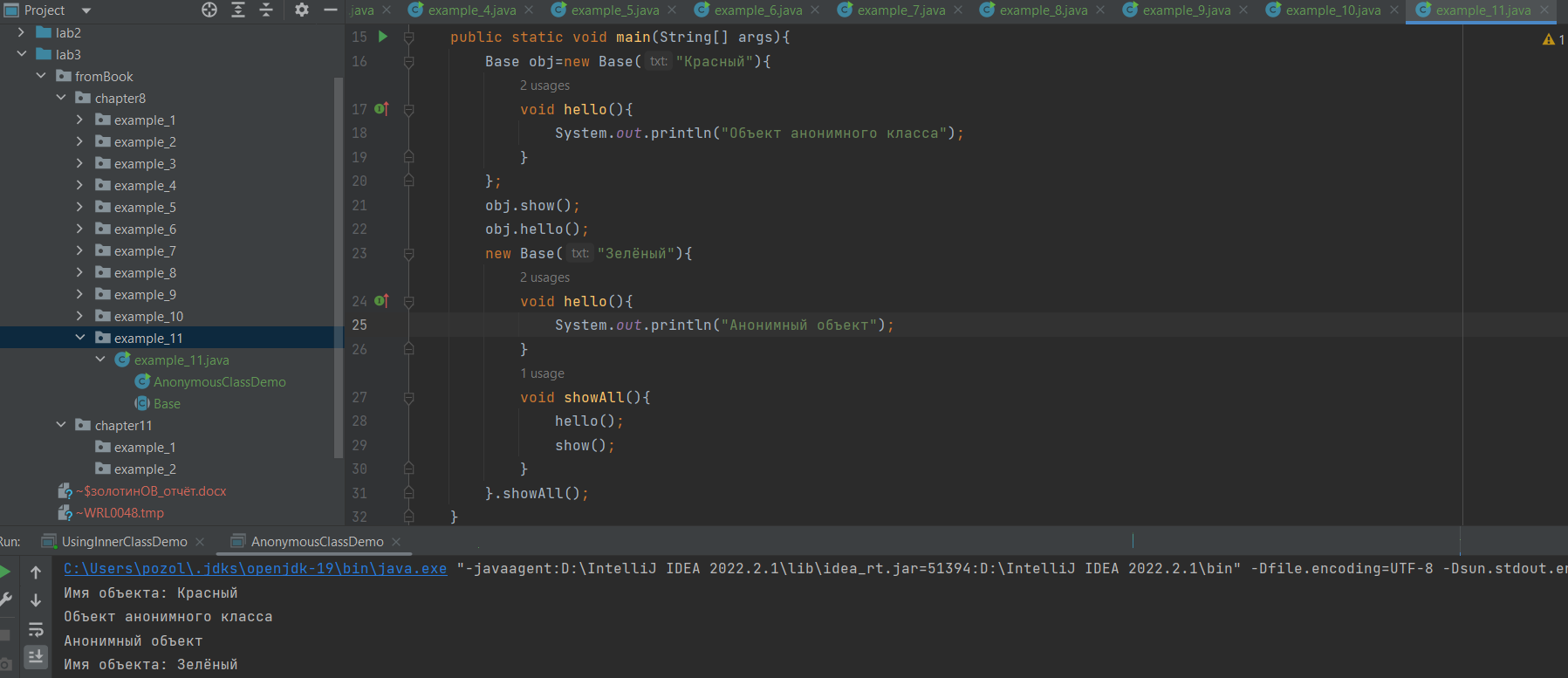
Анализ программного кода начнем с внутреннего класса Person, описанного в классе BankAccount. Класс Person достаточно простой. В нем есть:

текстовое поле name для запоминания имени вкладчика; • поле money типа double для записи значения начальной суммы вклада; • поле time типа int для записи значения интервала времени (в годах), на которое размещается вклад; • конструктор с тремя аргументами; • метод getMoney() для определения итоговой суммы по вкладу; • в классе переопределяется метод toString(). Особенность кода класса в том, что в методах getMoney() и toString() используется ссылка rate (процентная ставка) на поле внешнего класса BankAccount. Что касается класса BankAccount, то в нем кроме поля rate и внутреннего класса Person описан конструктор с четырьмя аргументами и метод show(), предназначенный для отображения информации по вкладу. Чтобы понять «взаимодействие» внутреннего и внешнего классов, имеет смысл проанализировать процесс создания объекта класса BankAccount (например, объекта ivanov в главном методе программы).

****

**Пример 11**

Небольшой пример, в котором использованы объекты, созданные на основе анонимного класса, который, в свою очередь, создается на основе абстрактного суперкласса, приведен в листинге 8.11.

****

**Пример 12**

Еще один часто используемый способ создания объектов анонимного класса базируется на реализации в анонимном классе интерфейса. Подход здесь абсолютно такой же, как и в случае с наследованием класса — с поправкой на то, что имеем дело с интерфейсом .

**Изображение выглядит как текст

Автоматически созданное описание**

**Резюме:**

• При передаче аргументов методам на самом деле передаются копии аргументов, которые создаются автоматически. Поэтому в теле метода нельзя изменить значение аргумента.

• При передаче аргументом методу объекта на самом деле передается ссылка на объект. Если нужно, чтобы метод возвращал объект результатом, то в теле метода создается соответствующий объект, а ссылка на него возвращается результатом метода.

• Если необходимо описать класс с полем-массивом, то полем является переменная массива, значением которой присваивается ссылка на массив. Массив обычно создается при вызове конструктора. Аналогичным образом поступают, если необходимо описать класс с полем-объектом: полем является объектная переменная, а значением ей присваивается ссылка на объект.

• Для создания массива объектов создается массив из объектных переменных, значениями которым присваиваются ссылки на объекты.

• Класс может быть описан в классе. Такой класс называется внутренним. Внутренний класс имеет доступ к полям внешнего класса.

• Объекты могут создаваться на основе анонимных классов. Такие анонимные классы создают, как правило, или наследованием абстрактного класса, или реализацией интерфейса.

**Глава 11**

**Пример 1**

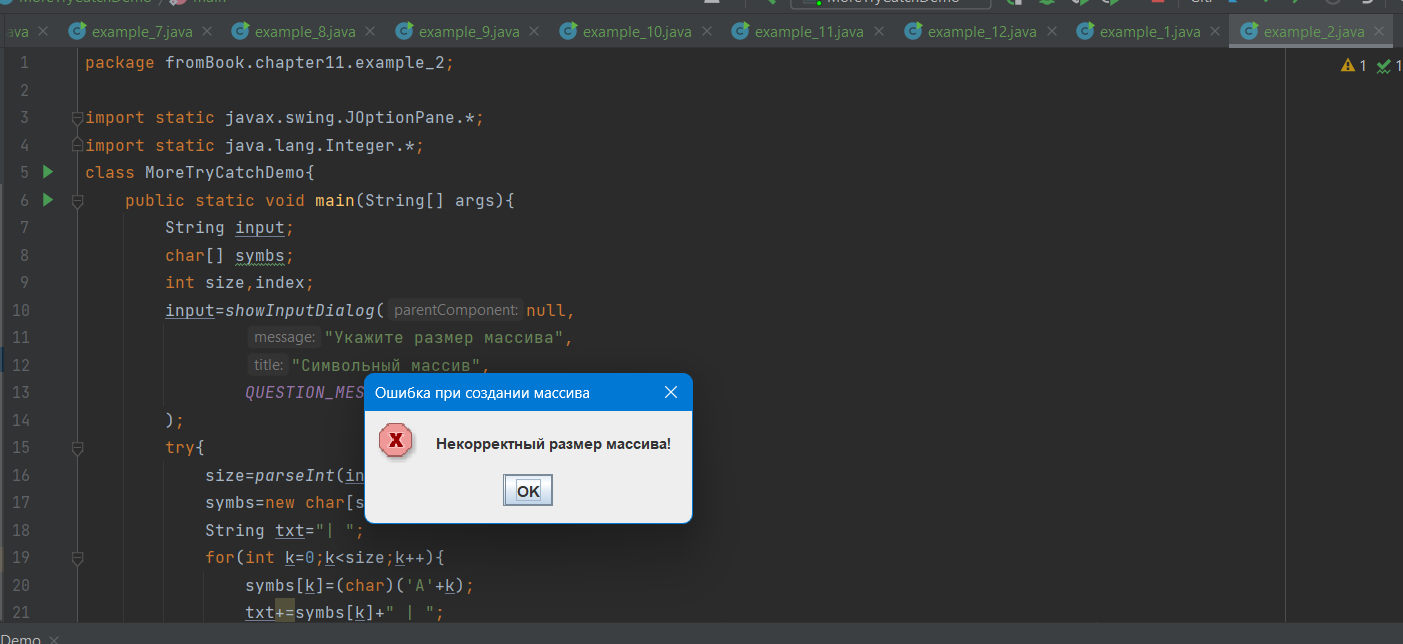
Рассмотрим программный код, представленный в листинге 11.1. Назначение у представленной там программы простое: отображается окно с полем ввода, в которое пользователя просят ввести целое число. Число считывается и в новом диалоговом окне отображается три числа: то, которое ввел пользователь, и еще два соседних числа.

**Изображение выглядит как текст, снимок экрана, телефон

Автоматически созданное описание**

**Пример 2**

Небольшой пример, в котором обрабатываются разные типы ошибок, представлен в листинге 11.2. В представленной программе отображается диалоговое окно, в котором пользователю предлагается указать размер числового массива. Если пользователь вводит корректное значение, то создается соответствующий числовой массив и заполняется символами. Последовательность символов из массива отображается в диалоговом окне. После этого пользователя просят указать индекс элемента в массиве, и после считывания введенного пользователем индекса отображается соответствующий символ. Также в программе отслеживаются возможные ошибки, связанные с тем, что: • пользователь отменил вод числа или ввел нечисловое значение; • ввел отрицательное значение для размера массива; • указал индекс элемента массива, который выходит за допустимый диапазон значений индекса.

****

**Пример 3**

Блоки try-catch могут быть вложенными. В таком случае если ошибка возникает во внутреннем try-блоке, то для обработки ошибки сначала просматриваются catch-блоки внутреннего try-блока (в котором возникла ошибка). Если ошибка внутренними catch-блоками не обрабатывается, то исключение передается для обработки во внешних catch-блоках. Далее рассматривается небольшая иллюстрация к использованию вложенных блоков обработки исключительных ситуаций. Задача, по большому счету, решается та же, что и в предыдущем примере, но способ решения другой. Во-первых, мы используем консольный ввод/вывод. Во-вторых, в программе использованы вложенные try-блоки. Рассмотрим программный код в листинге 11.3.

**Изображение выглядит как текст

Автоматически созданное описание**

**Пример 4**

В описании catch-блоков неизменно присутствует, кроме имени класса обрабатываемого в блоке исключения, еще и формальное обозначение для объекта исключения . Эти объекты можно использовать при обработке ошибок. В частности, объект исключения можно использовать в выражениях в качестве «текстового» операнда: благодаря переопределенному методу toString() объект исключения в таких случаях автоматически приводится к текстовому формату. Полученное текстовое значение содержит краткую, но все же полезную информацию о возникшей ошибке (в принципе это название класса ошибки и обычно минимальные поясняющие фразы). Очень простой пример представлен в листинге 11.4

**Изображение выглядит как текст

Автоматически созданное описание**

**Пример 5**

Хотя на первый взгляд это и выглядит несколько странным, но исключения можно генерировать , так сказать, «вручную». В частности, для искусственного генерирования исключений используется оператор throw , после которого указывается объект исключения. Объект исключения создают специально (как создается любой другой объект), или использую уже сгенерированный объект, переданный для обработки в catchблок. Пример, в котором используется искусственное генерирование исключений, представлен в листинге 11.5.

**Изображение выглядит как текст

Автоматически созданное описание**

**Пример 6**

Все исключения в Java делятся на контролируемые и неконтролируемые. К неконтролируемым исключениям относятся исключения классов, являющихся подклассами класса RuntimeException или класса Error . Все остальные исключения относятся к контролируемым. Разница между исключениями разных типов состоит в том, что для контролируемых исключений автоматически выполняется проверка на наличие обработки исключения. Другими словами, если метод может сгенерировать в процессе выполнения контролируемое исключение, то для такого исключения в методе должна быть предусмотрена обработка. Если метод все же не содержит код для обработки контролируемого исключения (но потенциально может такое исключение сгенерировать), то в описании метода в явном виде указывается, что он может генерировать необрабатываемое контролируемое исключение. Делается это просто: в описании метода после его имени и списка аргументов (но перед фигурными скобками с кодом метода) указывается ключевое слово throws и через запятую перечисляются классы контролируемых исключений, которые метод может сгенерировать, но которые в теле метода не обрабатываются. Некоторые классы контролируемых исключений представлены в табл. 11.2. Для большей наглядности рассмотрим несложный программный код, представленный в листинге 11.6.

**Изображение выглядит как текст

Автоматически созданное описание**

**Пример 7**

Существует возможность создавать пользовательские классы для исключений . Рецепт простой: на основе одного из классов исключений путем наследования создается класс. Объект, созданный на основе такого класса, может использоваться при генерировании исключительных ситуаций. Например, в качестве суперкласса можем использовать класс Exception , или, скажем, класс RuntimeException . В первом случае получим пользовательский класс исключения контролируемого типа, а во втором — неконтролируемого типа. Небольшая иллюстрация к созданию и использованию пользовательских классов для исключений представлена в листинге 11.7.

Изображение выглядит как текст

Автоматически созданное описание

**Резюме**

• Для обработки ошибок (исключений), возникающих в процессе выполнения программы, используется конструкция try-catch. Контролируемый код помещается в try-блок, а для обработки возможных ошибок используются catch-блоки.

• Каждому типу ошибки соответствует определенный класс. Классы исключений образуют иерархию наследования. Наиболее важные классы: Exception, Error, RuntimeException, и ряд других.

• Если при выполнении кода в try-блоке возникает ошибка, то выполнение кода прекращается, автоматически создается объект исключения, и он передается для обработки в catch-блок, соответствующий ошибке данного типа.

• В каждом catch-блоке указывается названия класса для обрабатываемого исключения и название для объекта исключения. В catch-блоке обрабатываются не только сообщения указанного в инструкции класса, но и исключения подклассов данного класса. После выполнения команд в catch-блоке начинает выполняться следующая после конструкции try-catch команда. Если при выполнении try-блока ошибки не возникли, то catch-блоки игнорируются. Также в try-catch инструкции может использоваться блок finally. Команды в finally-блоке выполняются в любом случае — и при возникновении ошибки, и при отсутствии ошибок.

• Для искусственного генерирования исключения используют оператор throw, после которого указывается объект исключения.

• Все исключения делятся на контролируемые и неконтролируемые. Неконтролируемым исключениям соответствуют классы исключений, являющиеся наследниками классов RuntimeException и Error. Если метод может выбрасывать контролируемое исключение, которое не обрабатывается в методе, то в названии метода указывается инструкция throws, после которой указывается класс выбрасываемого исключения.

• Можно создавать пользовательские классы исключений. Пользовательский класс исключения создается путем наследования одного из классов исключений.