Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра «Программное обеспечение информационных систем

и технологий»

**Отчёт**

по лабораторной работе №2

по дисциплине ***«Программирование на языке Java»***

Выполнила**:** студентка группы *10702121*

Алюнина П.А.

Проверил**:**  Сидорик В. В.

Минск – 2023

**Цель:** Ознакомление с операторами сравнения if-else, операторами выбора switch. Изучение операторов цикла: for, do-while, while. Работа с операторами break, continue. Использования меток с именами – label. Просмотр возможностей класса Math.

**Задачи лабораторной работы:**

1. Изучить и законспектировать теорию
2. Повторить выполнения примеров заданий
3. Выполнить задания с индивидуальными данными
4. Оформить отчёт с теорией и скриншотами индивидуальных заданий
5. Составить глоссарий
6. Ответить на контрольные вопросы

# **Задание №1 IfTestSmall**

Изучите и законспектируйте теорию:

1. If – оператор

Оператор **if** начинается с ключевого слова **if**. Ключевое слово **if** должно сопровождаться булевым выражением, заключённым в скобки. Самая простая форма выглядит так:

**if (условие) оператор; // если условие истинно, то выполняется оператор**

Здесь условие — это булево выражение, имеющее значение *true* или *false*. Если условие истинно, то оператор или группа операторов выполняется, если ложно, то оператор не выполняется.

Оператор **if** продолжается заключенным в фигурные скобки фрагментом, который называют блоком операторов. Если используется только один оператор, то фигурные скобки можно опустить.

Существует расширенный вариант оператора **if** с использованием ключевого слова **else**:

**if (условие) оператор;// если условие истинно, то выполняется первый оператор**

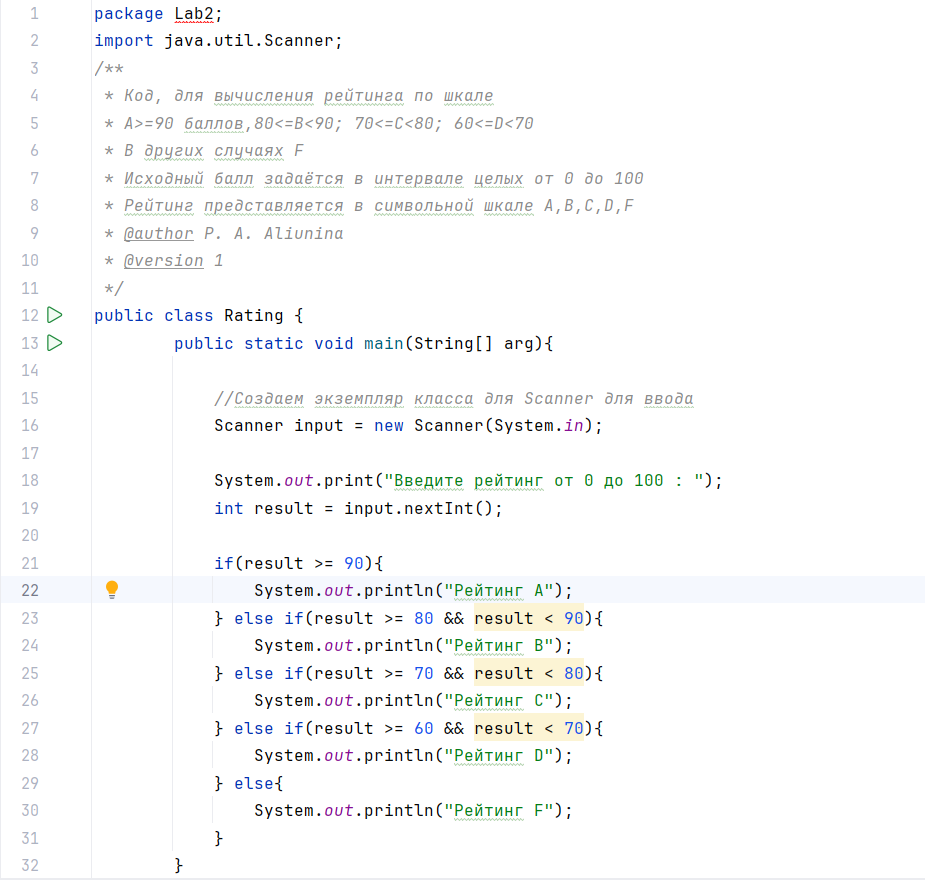
**else оператор;// если условие ложно, то выполняется оператор после else**

В этом случае при выполнении условия оператора **if** инициируется только один оператор, если условие не выполняется, то также инициируется только один оператор, который относится к **else**. Также можно использовать блоки операторов.

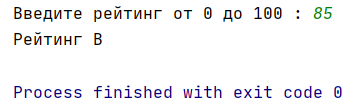
Самостоятельная работа:

Написать код для вычисления рейтинга по шкале: A >= 90 баллов, 80<=B<90, 70<=C<80, 60<=D<70. В других случаях - D. Исходный балл задается в интервале целых от 0 до 100. Рейтинг представляется в символьной шкале A, B, C, D, F. Данные в баллах вводить с консоли. Результат вывести на печать. Используйте конструкцию if ... else if ... else if ... else. Код оформить в виде отдельного класса Rating

1. **Код программы**



1. **Скриншот выполнения программы**



**Ответы на контрольные вопросы**

1) В каких случаях используется if оператор?

Когда для продолжения работы программы необходима проверка какого-либо условия.

**Задание №2 ForLoopTest**

Изучите и законспектируйте теорию:

1. For - оператор

Конструкция **for** управляет циклами. Команда выполняется до тех пор, пока управляющее логическое выражение не станет ложным. Цикл **for** проводит инициализацию перед первым шагом цикла. Затем выполняется проверка условия цикла, и в конце каждой итерации происходит изменение управляющей переменной.

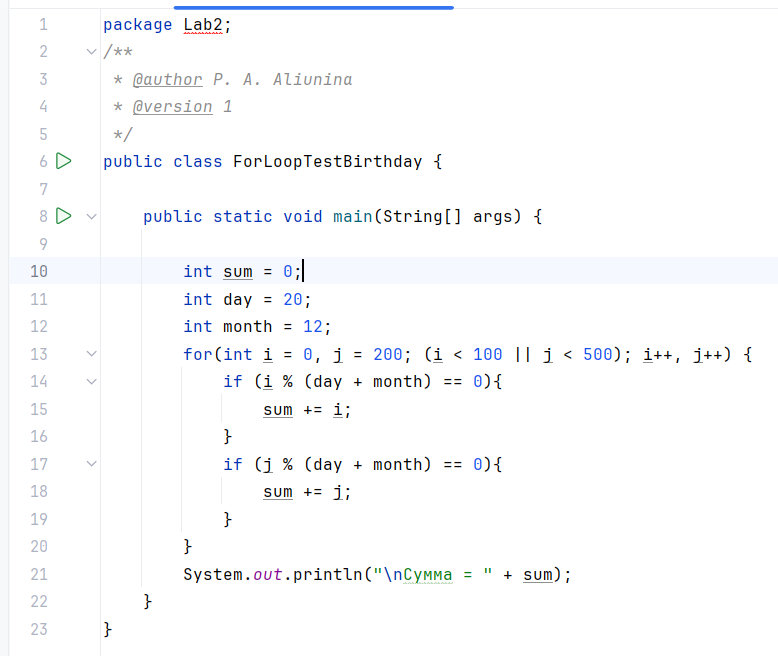
**for (инициализация; логическое выражение (условие); шаг (итерация)) команда;**

Любое из трёх выражений цикла (инициализация, логическое выражение или шаг) можно пропустить. Перед выполнением каждого шага цикла проверяется условие цикла. Если условие окажется ложным, то выполнение продолжится с инструкции, следующей за конструкцией **for**.

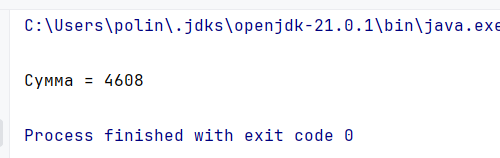
Самостоятельная работа:

Используя оператор for напишите код вычисления суммы целых чисел, которые делятся на день + месяц вашего рождения без остатка по двум переменным одновременно: от 0 до 100 и от 200 до 500. Код оформить в виде отдельного класса.

1. **Код программы**



**2) Скриншоты выполнения программы**



**Ответы на контрольные вопросы**

1. В каких случаях используется for оператор?

1) Когда необходимо выполнить однообразное действие множество раз.

2) Когда операция выполняется над переменной и при каждой итерации необходимо её увеличивать (или уменьшать) на определённую величину.

2. Как организовать цикл по двум переменным?

int n = 8;  
for (int i=0, int j = n - 1; i < j; i++, j--) {  
System.out.println(i \* j);  
}

**Задание № 3 DoWhileTest**

Изучите и законспектируйте теорию:

1. do-while - оператор

do  
// команда (тело цикла)

while (условие-логическое выражение)

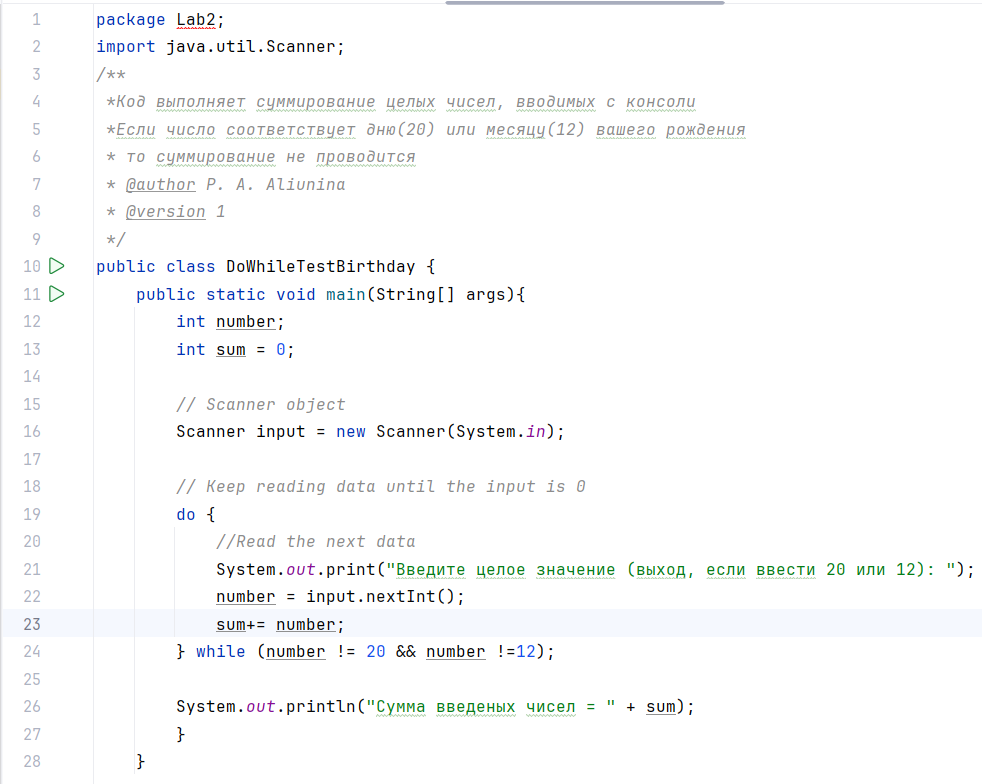
Отличие цикла **do-while** от цикла while состоит в том, что цикл **do-while** выполняется по крайней мере один раз, даже если условие изначально ложно. В цикле **while** такое не произойдёт, так как тело цикла не отработается. Цикл **do-while** используется реже, чем **while**.

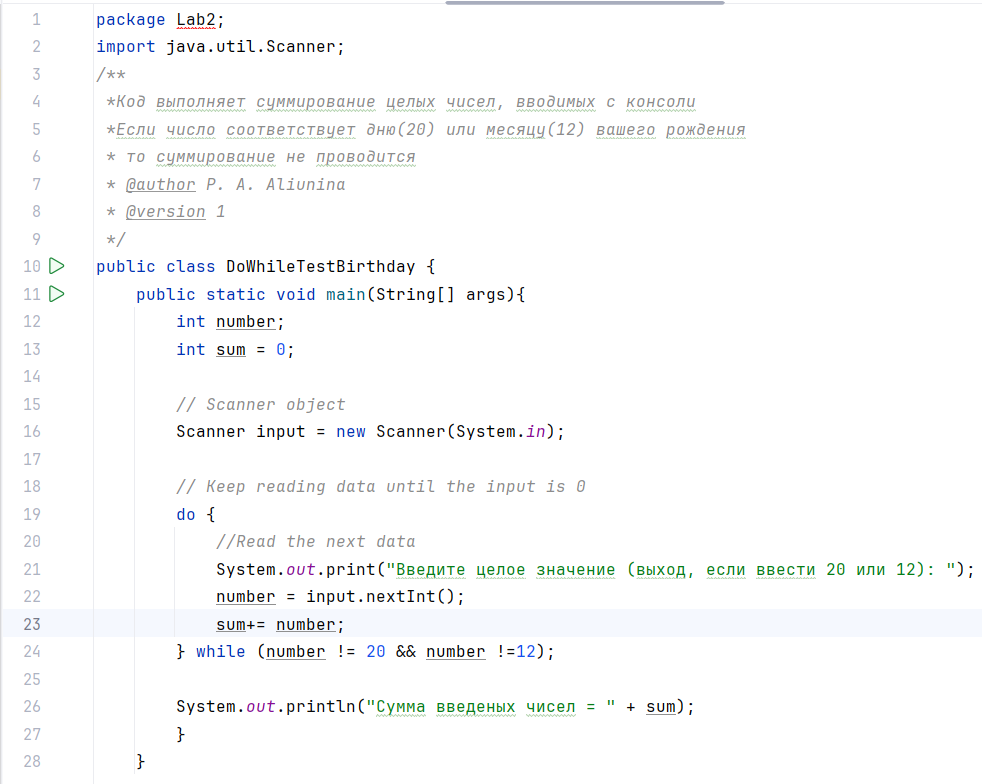
Бывают ситуации, когда проверку прерывания цикла желательно выполнять в конце цикла, а не в его начале. И данный цикл позволяет это сделать. При каждом повторении цикла **do-while** программа сначала выполняет тело цикла, а затем вычисляет условное выражение. Если это выражение истинно, то цикл повторяется. В противном случае выполнение цикла прерывается. Как и в других циклах Java, условие должно иметь булево значение.

Самостоятельная работа:

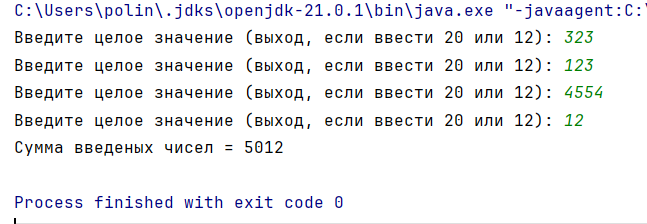
Используя оператор do...while напишите код вычисления суммы целых чисел, вводимых с консоли. Если число соответствует дню или месяцу вашего рождения – суммирование не проводится. Код оформить в виде отдельного класса.

1. **Код программы**

****

****

**2) Скриншоты выполнения программы**



**Ответы на контрольные вопросы**

1. В каких случаях используется do-while оператор?

Когда проверка условия необходима после выполнения цикла.

2. Что означает data! = 0 в коде программы при выполнении?

Значение переменной data не равно нулю.

**Задание № 4 WhileTest**

Изучите и законспектируйте теорию:

1. Оператор while и его использование

**while(условие) {**// тело цикла

**}**

Здесь *условие* должно быть любым булевым выражением. Тело цикла будет выполняться до тех пор, пока условное выражение истинно. Когда *условие* становится ложным, управление передаётся строке кода, которая идёт после цикла. Если в цикле используется только один оператор, то фигурные скобки можно опустить.

Поскольку цикл **while** вычисляет своё условное выражение в начале цикла, то тело цикла не будет выполняться, если условие с самого начала было ложным.

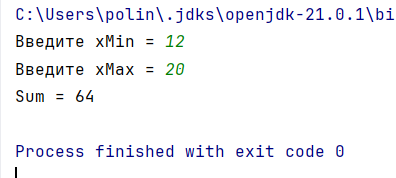
Самостоятельная работа:

Используя оператор while напишите код вычисления суммы нечетных (целых) чисел от xMin до Xmax. В качестве xMin и Xmax взять день и месяц вашего рождения. Ввод данных осуществлять с консоли. Если они одинаковы, для получения максимального одно из чисел умножить на десять. xMin, Xmax и результат вывести на печать. Код оформить в виде отдельного класса.

1. **Код программы**



**2) Скриншоты выполнения программы**



**Ответы на контрольные вопросы**

1. В каких случаях используется while и в чем отличие от for?

While используется когда неизвестно количество итераций цикла.

2. Если i=3, то каков результат sum += i; ? Объясните.

Суммирование в примере начнётся с 3-х до xMax.

sum = 3 + 4 + … + xMax;

**Задание №5 BreakTest**

Изучите и законспектируйте теорию:

1. Break оператор

Оператор break позволяет выйти из цикла в любой его момент, даже если цикл не закончил свою работу.

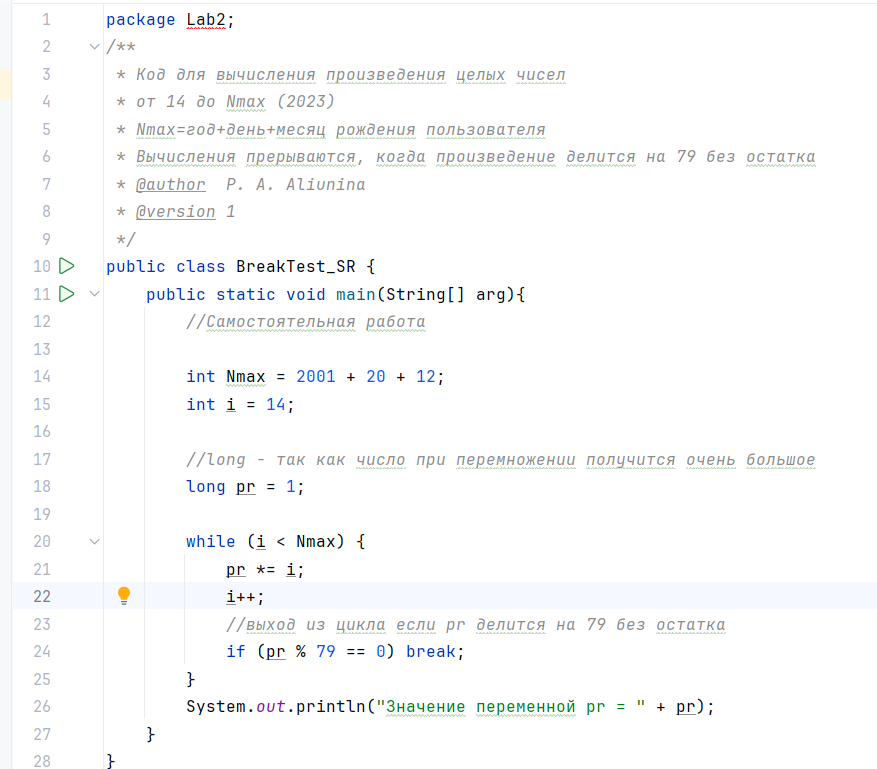
Пример:

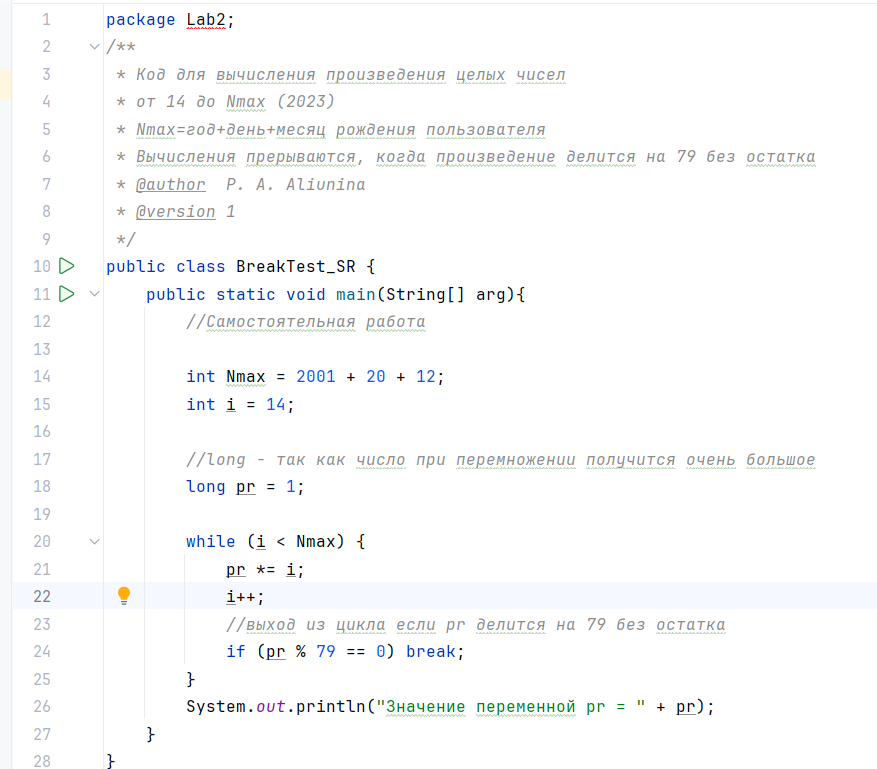
while(I < Nmax){  
 num \*= i;  
 i++;  
 if ( (num % 25) == 0) break;  
 }

Самостоятельная работа:

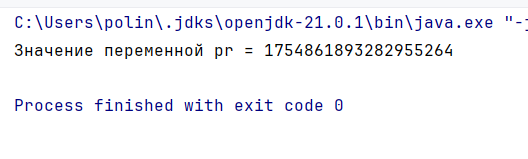
Напишите код вычисления произведений целых чисел от 14 до Nmax. В качестве Nmax возьмите год+день+месяц вашего рождения. Вычисления прервите, как только произведение будет делиться на 79 без остатка. Код оформить в виде отдельного класса.

1. **Код программы**

****

****

**2) Скриншоты выполнения программы**



**Ответы на контрольные вопросы**

1) В каких случаях используется break оператор?

Когда нам при выполнении некоторого условия необходим досрочный выход из цикла.

**Задание № 6 ContinueTest**

Изучите и законспектируйте теорию:

1. continue - оператор

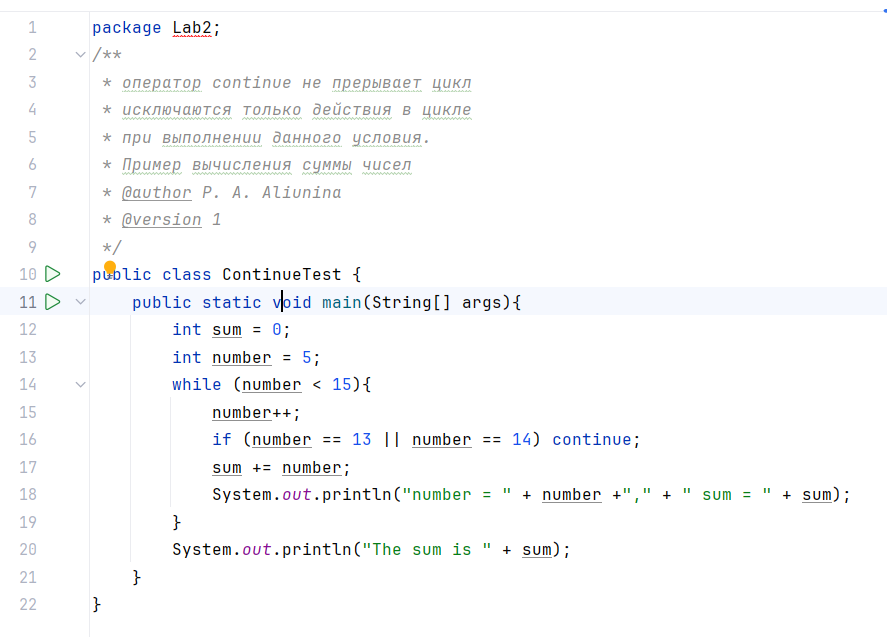
Оператор continue позволяет пропустить часть цикла, которая находится после него, то есть перенаправляет в начало цикла, а не выходит из него (в отличие от оператора break)

Пример

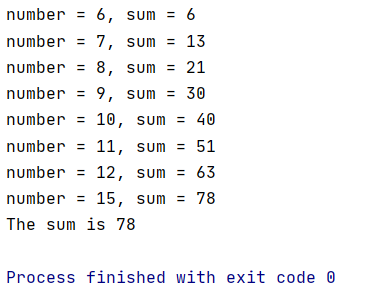
while (number < 30) {  
 number++;  
 if (number == 5 || number == 28) continue;  
 sum += number;  
 System.out.println("number=" + number + "sum=" + sum);  
 }

То есть, если условие в операторе if будет истинным, то последние две строчки цикла не выполнятся.

1. **Код программы**



**2) Скриншот выполнения программы**



**Ответы на контрольные вопросы**

1) Опишите логику применения оператора continue в циклах?

Оператор continue используется в циклах, когда при выполнении некоторого условия необходимо пропустить часть цикла, которая находится за оператором continue, но полный выход из цикла при этом не нужен.

**Задание №7 SwitchTest**

Изучите и законспектируйте теорию:

1. Оператор switch и его использование

В отличие от операторов **if** и **if -else**, оператор **switch** применим к известному числу возможных ситуаций.

Дублирование значений **case** не допускается. Тип каждого значения должен быть совместим с типом выражения.

Команду **switch** часто называют командой выбора. Выбор осуществляется в зависимости от целочисленного выражения. Форма команды выглядит так:

switch (Выражение\_для\_сравнения) {  
 case Совпадение 1:  
 команда;  
 break;  
 case Совпадение 2:  
 команда;  
 break;  
 case Совпадение 3:  
 команда;  
 break;  
default:  
 оператор;  
 break;  
 }

Параметр *Выражение Для Сравнения* - выражение, в результате вычисления которого получается целое число (как правило).

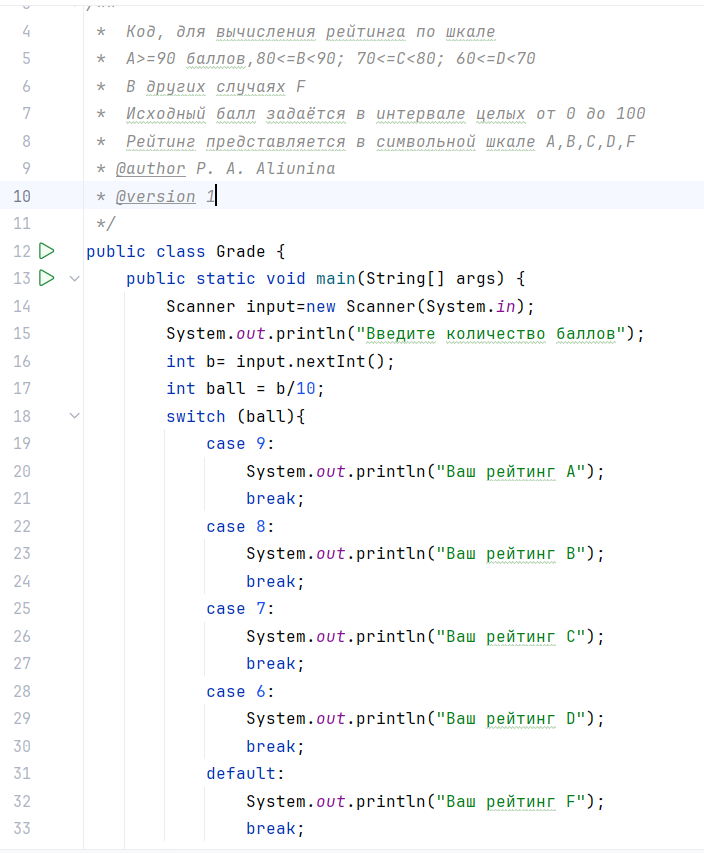
Команда **switch** сравнивает результат *Выражение* Для *Сравнения* с каждым последующим *Совпадением*. Если обнаруживается совпадение, то исполняется команда или набор команд, которые прописаны за данным оператором. Если совпадений не будет, то исполняется команда после ключевого слова **default**. Однако оператор **default** не является обязательным. В этом случае при отсутствии совпадений программа не выполняет никаких действий.

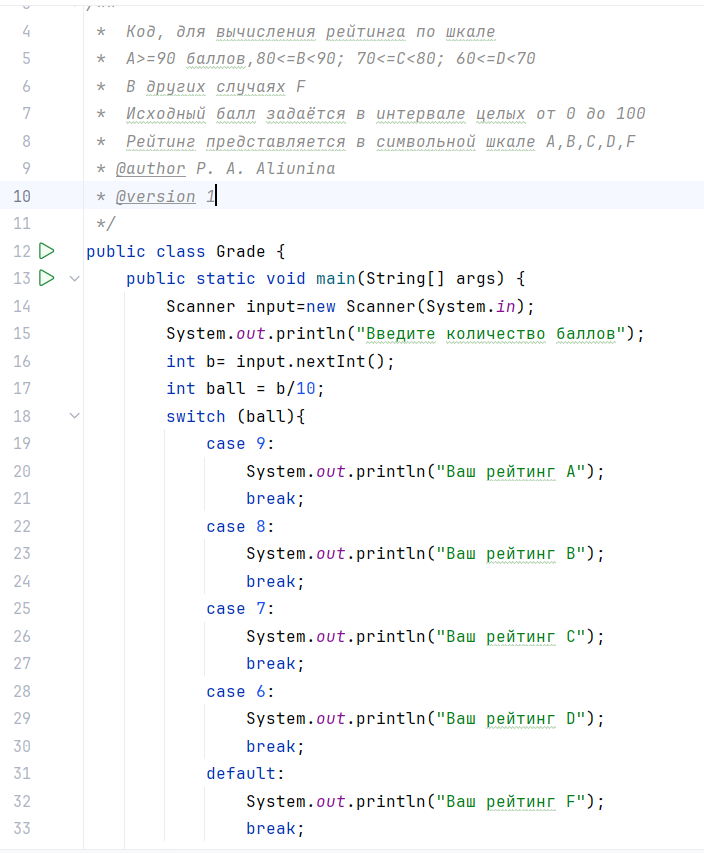
Каждая секция **case** обычно заканчивается командой **break**, которая передаёт управление к концу команды **switch**.

Самостоятельная работа:

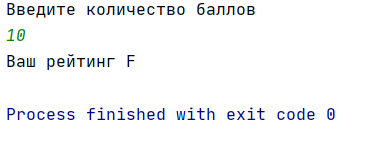
Используя оператор Switch написать код для вычисления рейтинга по шкале: 90<=A<100, 80<=B<90, 70<=C<80, 60<=D<70. В других случаях - F. Исходный балл задается в интервале целых от 0 до 100. Рейтинг представляется в символьной шкале A, B, C, D, F. Данные в баллах вводить с консоли. Результат вывести на печать. Маленькая хитрость: разделите балл на 10 (для анализа в switch). Код оформить в виде отдельного класса Grade.

1. **Код программы**

****

****

**2) Скриншоты выполнения программы**



**Ответы на контрольные вопросы**

1. Какова роль break при использовании switch?

Оператор break предназначен для выхода из цикла, то есть при применении break в каждом блоке case будет происходить следующая ситуации: если переменная x в switch (x) совпадает со значением в case, то выполняется команда, которая прописана в этом case, и если после команды (или блока команд) отсутствует оператор break, то выполнение switch продолжится (будут проверятся остальные case), а это бессмысленная трата ресурсов.

2. Что такое графическая библиотека Swing и как ее подключить?

Для создания графического интерфейса приложения необходимо использовать специальные компоненты библиотеки Swing, называемые контейнерами высшего уровня (top level containers). Они представляют собой окна операционной системы, в которых размещаются компоненты пользовательского интерфейса. К контейнерам высшего уровня относятся окна JFrame и JWindow, диалоговое окно JDialog, а также апплет JApplet (который не является окном, но тоже предназначен для вывода интерфейса в браузере, запускающем этот апплет).

Подключить библиотеку можно в начале программы написав:

import javax.swing.\*;// подключаем графическую библиотеку Swing

**Задание №8 LabelTest**

Изучите и законспектируйте теорию:

1. label и continue

**label** — имя метки, которая идентифицирует некоторый блок кода. Когда эта форма break выполняется, управление передается из именованного блока кода (чья метка указана в операторе break) на следующий за этим блоком оператор. Помеченный блок кода обязательно должен включать данный оператор break, но не требуется, чтобы это включение было непосредственным (т. е. break может включаться не прямо в блок со своей меткой, а во вложенный в него блок, возможно, тоже помеченный). Это означает, что вы можете использовать помеченный оператор break, чтобы выйти из набора вложенных блоков. Но вы не имеете возможности использовать break для передачи управления блоку кода, который не включает оператор break.

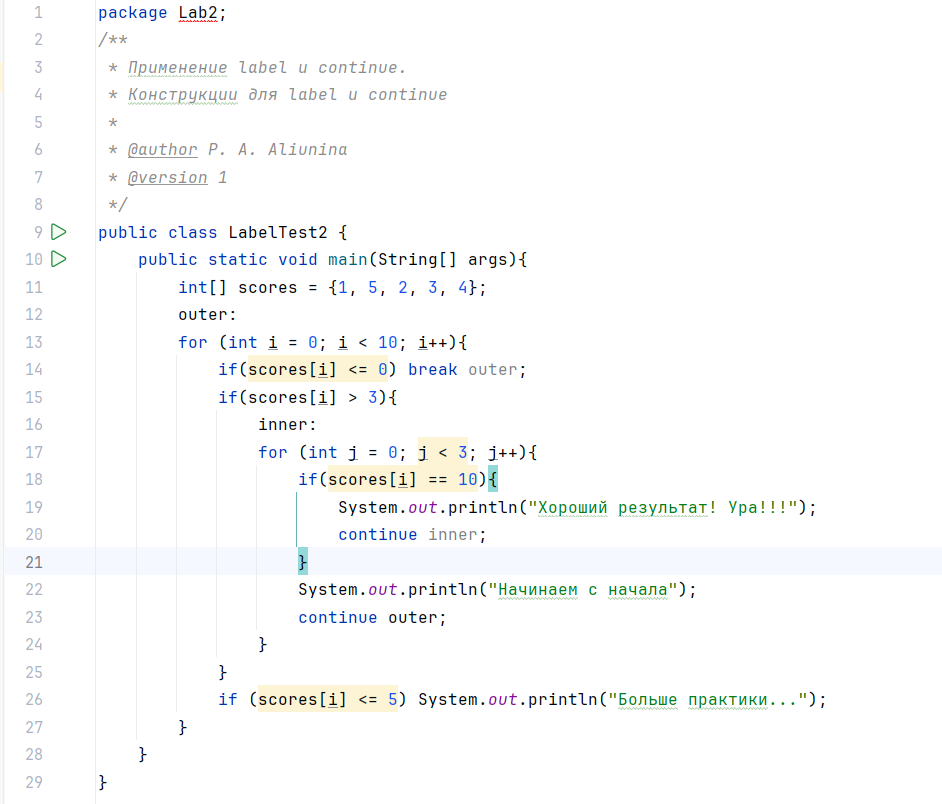
public class ContinueLabel {  
  
 public static void main(String[] args) {  
 outer:  
 for (int i = 0; i < 10; i++) {  
 for (int j = 0; j < 10; j++) {  
 if (j > i) {  
 System.*out*.println();  
 continue outer;  
 }  
 System.*out*.print(" " + (i \* j));  
 }  
 }  
 System.*out*.println();  
 }  
}

Оператор continue в этом примере завершает цикл, вычисляющий j, и продолжает со следующей итерации цикла, управляемого i.

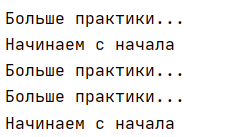
Самостоятельная работа

Измените массив исходных данных. Добавьте вывод на печать (консоль) значений, которые оцениваются. Добавьте оценку значения меньше трех с сообщением "Начинаем с начала".

1. **Код программы**

****

**Скриншот выполнения программы**

****

**Ответы на контрольные вопросы**

1. В каких случаях используется label?

Когда нам необходим выход из блока кода. Это блок кода можно сделать именованным с помощью label. При использовании конструкции break label управление передаётся на следующий за этим блоком оператор. А при использовании continue label можно определить, какой именованный блок кода следует продолжить.

2. Какие метки используются в данной программе.

Outer, inner.

**Задание № 9 MathClassExample**

Изучите и законспектируйте теорию:

1. Класс Math и его использование

Класс **Math** содержит методы, связанные с геометрией и тригонометрией и прочей математики. Методы реализованы как **static**, поэтому можно сразу вызывать через **Math.methodName()** без создания экземпляра класса.

В классе определены две константы типа *double*: **E** и **PI**.

Методов множество, и интересующий можно найти в документации о классе Math.

Самостоятельная работа:

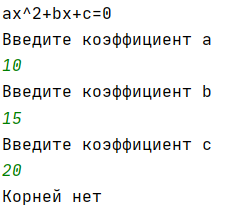
Вычислить корни квадратного уравнения х1 и х2. Параметры a, b, c - задать вводом с консоли. a, b, c и корни вывести на печать. Формула для расчета стандартная. Программу оформите в виде отдельного класса.

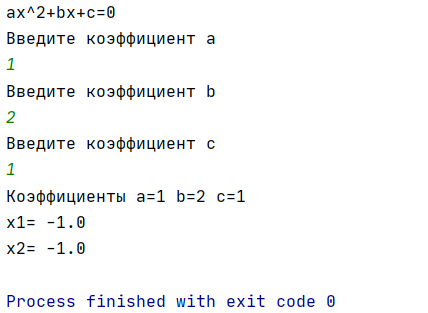
1. **Код программы**

****

****

**2) Скриншоты выполнения программы**





**Ответы на контрольные вопросы**

1. Почему класс Math не импортируется?

Класс Math содержит методы, связанные с геометрией и тригонометрией и прочей математики. Методы реализованы как static, поэтому можно сразу вызывать через Math.methodName() без создания экземпляра класса.

Глоссарий

**Операторы**— это специальные символы, которые сообщают транслятору о том, что вы хотите выполнить операцию с некоторыми операндам

**Опера́нд** ― аргумент операции; данные, которые обрабатываются командой; грамматическая конструкция, обозначающая выражение, задающее значение аргумента операции; иногда операндом называют место, позицию в тексте, где должен стоять аргумент операции.

**Цикл**— разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций.

**Компиля́тор** – это программа, которая переводит текст, написанный на языке программирования, в набор машинных кодов.

**Компиля́ция** — сборка программы, включающая [трансляцию](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80" \o "Транслятор) всех модулей программы, написанных на одном или нескольких исходных [языках программирования высокого уровня](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F" \o "Высокоуровневый язык программирования) и/или [языке ассемблера](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0" \o "Язык ассемблера), в эквивалентные программные модули на [низкоуровневом языке](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F" \o "Низкоуровневый язык программирования), близком [машинному коду](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4" \o "Машинный код) ([абсолютный код](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D0%BE%D0%BB%D1%8E%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4" \o "Абсолютный код), [объектный модуль](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C" \o "Объектный модуль), иногда на [язык ассемблера](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0" \o "Язык ассемблера)) или непосредственно на машинном языке или ином [двоичнокодовом](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4" \o "Двоичный код) низкоуровневом командном языке и последующую сборку исполняемой машинной программы.

**Интерпрета́тор** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) *interpreter* [ıntə:'prıtə](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82" \o "Международный фонетический алфавит)[[1]](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80" \l "cite_note-1), от [лат.](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D1%82%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Латинский язык) *interpretator* - толкователь) — [программа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) (разновидность [транслятора](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80)), выполняющая *интерпретацию*.

**Интерпрета́ция** — построчный анализ, обработка и выполнение исходного кода программы или запроса (в отличие от [компиляции](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80), где весь текст программы, перед запуском, анализируется и транслируется в машинный или байт-код, без её выполнения) .

**Класс**— в объектно-ориентированном программировании, представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов.