Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа «Компьютерных технологий и информационных систем»

**ОТЧЕТ по лабораторной работе №5**

на тему “Коллекции в Java” по дисциплине

«Системный подход в разработке программного обеспечения»

**Выполнил:**

студент группы 5130902/20201 \_\_\_\_\_\_\_\_\_\_\_\_\_ А. И. Сафонов

подпись

**Проверил:**

Кандидат тех. наук, доцент \_\_\_\_\_\_\_\_\_\_\_\_ С. А. Нестеров

подпись

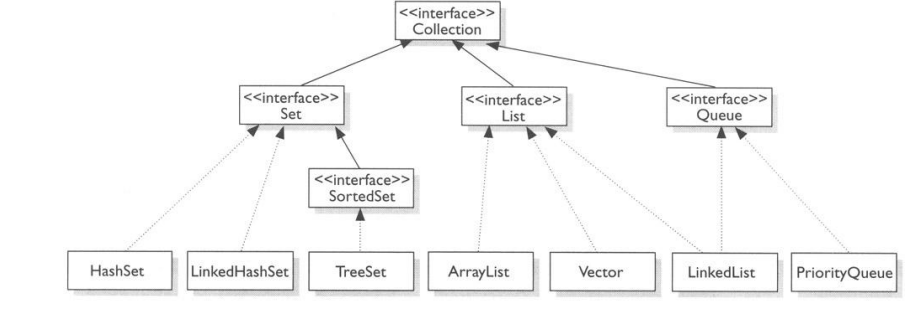
«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025г.

Санкт-Петербург, 2025

А.А. Дубаков ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА JAVA

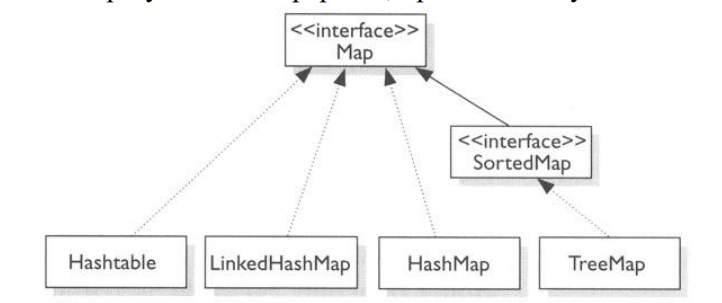
До сих пор нам известен только один способ хранения коллекции объектов - массивы Java, которые хороши для хранения, но проявляют недостатки, когда размер коллекции не фиксирован и необходимо динамически добавлять или удалять данные, или сортировать их.

Collection является корневым интерфейсом в иерархии, из которого производятся интерфейсы List, Queue и Set, реализующиеся в кассах на нижнем уровне иерархии, как представлено на Рис. 45.



* Set - коллекция набора, не содержащая дубликатов.
* List - упорядоченная коллекция списка, содержащего дубликаты.
* Queue - обычно коллекция очереди FIFO (первым пришел - первым обслужись - first-in, first-out), которая моделирует обслуживание в типичной очереди - каждый новый элемент добавляется в конец очереди, и элементы извлекаются с начала очереди. Могут быть определены другие порядки обслуживания (LIFO - стек (последним пришел - первым обслужись - lastin, first-out), которая извлекает из очереди последнего добавленного в нее.

Отображение Map не является потомком интерфейса Collection и образует свою иерархию, представленную на Рис. 46.

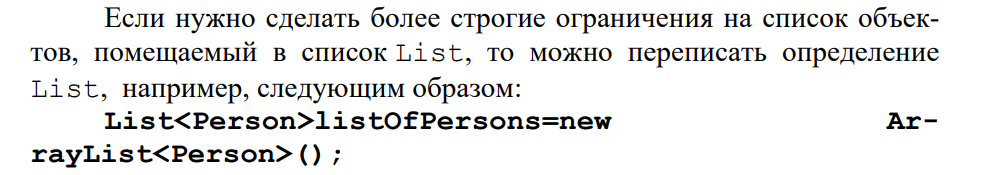


List – не является классом в привычном понимании (это интерфейс), поэтому его экземпляр нельзя создать непосредственно.



**Объявление формального типа**

< Object > - это формальный тип, который указывает компилятору на то, что список List содержит коллекцию типа Object, то есть в данном описании в List можно помещать любой объект, поскольку любой объект является потомком объекта



**Использование List**

* add([int index,] variable) - поместить некоторый объект в List;
* size() - узнать, какой размер списка List в текущий момент;
* get(int index) - извлечь объект из списка List.

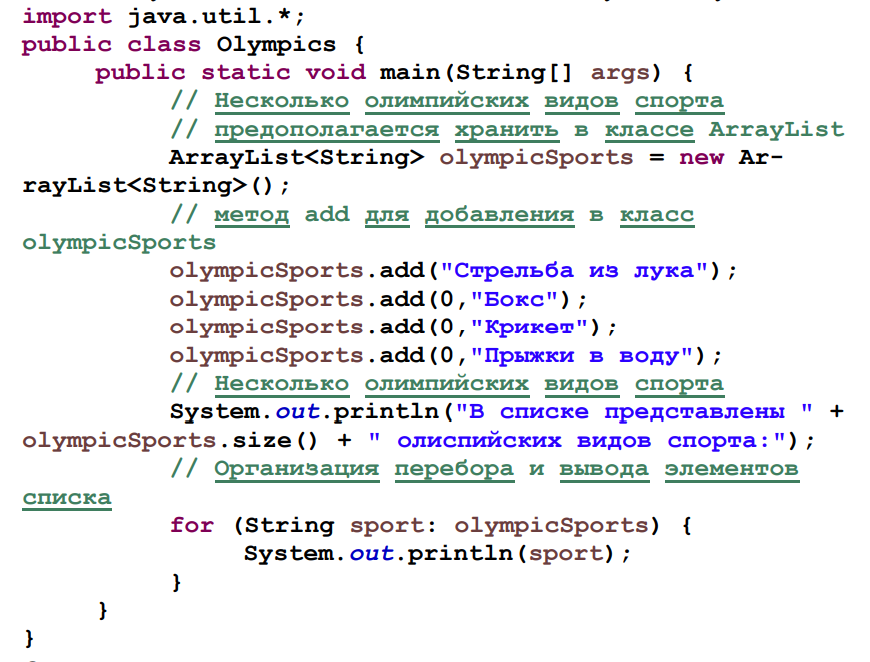
Интерфейс Iterable

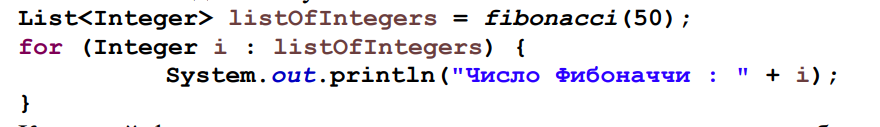
В интерфейсе Iterable описан только один метод: iterator(), который является конструктором объекта типа Iterator, т.е. объекта, который поочерёдно возвращает все элементы коллекции. Применительно к конкретной имеющейся коллекции, например, List listOfIntegers = new ArrayList(); итератор может описан следужщим образом: Iterator iterator = listOfIntegers.iterator()

Рассмотрим подробнее методы возвращаемого класса Iterator:

* boolean hasNext() - метод возвращает true, если в коллекции ещё остались элементы и false, если достигнут конец коллекции.
* next() - метод возвращает текущий элемент. Поскольку итератор настраивается на работу с объектами определенных классов, то этот метод нам будет возвращает не Object, а сразу тот тип, который содержится в списке.
* void remove()- метод удаляет из коллекции последний возвращенный итератором элемент. Этот метод может быть вызван только однократно на один вызов next().

Перебор элементов List



****

Set ‐ Набор

Набор (Set) – это коллекция, которая, по определению, содержит только уникальные элементы – в ней отсутствуют дубликаты объектов. Если List может содержать одни и те же объекты многократно, то Set может содержать данный объект только один раз. Java коллекция Set может содержать только объекты и строго регламентирует их поведение. Поскольку Set является интерфейсом, нельзя создать его экземпляр непосредственно, необходимо использовать одну из реализаций интерфейса Set: HashSet, LinkedHashSet или TreeSet.

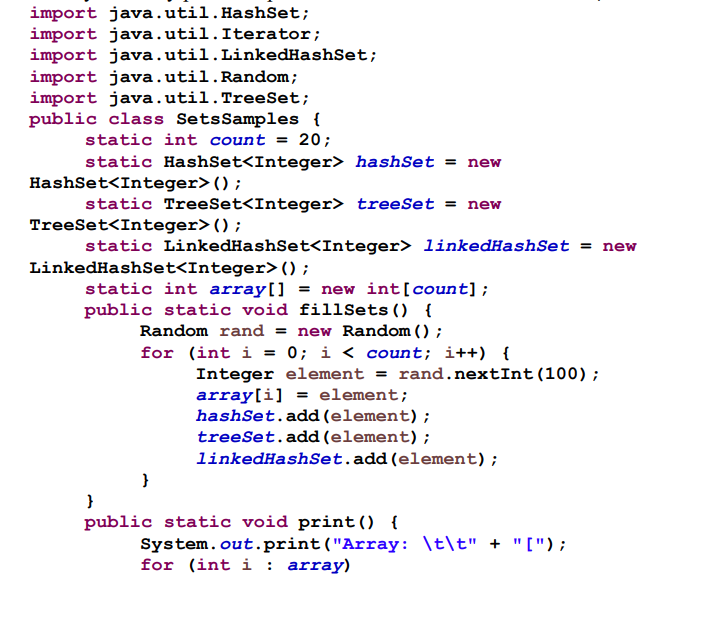
Главное отличие состоит в порядке хранения элементов. HashSet хранит элементы в случайном (на первый взгляд) порядке. Однако элементы внутри набора HashSet упорядочены в соответствии со значением hashCode, которое рассчитывается для каждого элемента. По значению hashCode также осуществляется поиск элемента в наборе. TreeSet, в отличие от HashSet, хранит элементы упорядоченно, то есть в какой бы последовательности не добавлялись и не удалялись элементы, коллекция останется строго упорядоченной. LinkedHashSet используется в том случае, если необходимо помнить порядок добавления элементом. Поиск по этой коллекции происходит также по значению hashCode, но порядок будет всегда совпадать с очерёдностью добавления.

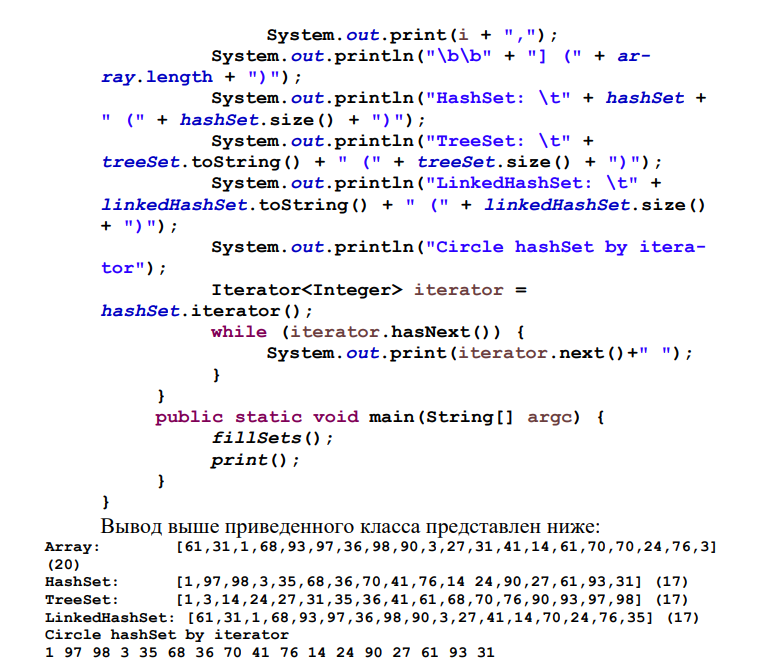
Рассмотрим некоторые методы, которые позволяет выполнять интерфейс Set: •

boolean add([index,] variable) - поместить некоторый объект variable в набор Set; возвращает true, если набор не содержит элемент. Можно указать порядковый номер элемента в наборе в переменной index;

* int size() - возвращает целое, указывающее какой размер списка Set в текущий момент;
* addAll(Collection c) - добавляет все элементы коллекции с (если их ещё нет); • clear() - удаляет все элементы коллекции;
* boolean contains(Object o) - возвращает true, если элемент есть в коллекции;
* boolean containsAll(Collection c) - возвращает true, если все элементы содержатся в коллекции
* boolean isEmpty() - возвращает true, если в коллекции нет ни одного элемента;
* boolean remove(Object o) - удаляет первое вхождение указанного элемента из этого списка, если он существует. Если список не содержит элемент, он остается неизменным.
* boolean removeAll(Collection c) - удаляет из этого набора все его элементы, которые содержатся в указанном наборе. Если указанная коллекция также набор, эта операция эффективно изменяет этот набор, так что его значение является ассиметричной разностью двух множеств;
* boolean retainAll(Collection c) - оставляет только элементы в этом наборе, которые содержатся в указанном наборе. Другими словами, удаляет из этого множества все элементы, которые не содержатся в указанном наборе. Если указанная коллекция также набор, эта операция эффективно изменяет этот набор, так что его значение является пересечением двух множеств;
* Object[] toArray() - возвращает массив, содержащий элементы коллекции.

Использование Set



****

Map ‐ Отображение

Интерфейс Map соотносит уникальные ключи со значениями. Ключ - это объект, который используется для последующего извлечения данных. Задавая ключ и значение, вы можете помещать значения в объект отображения. После того как это значение сохранено, вы можете получить его по ключу.

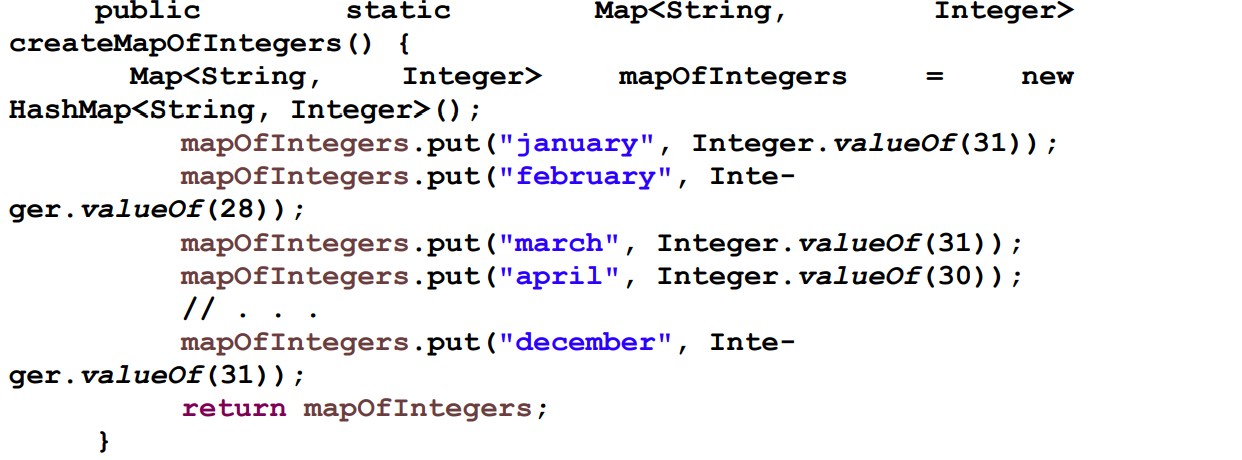
Отображение (Map) – это удобная структура, которая позволяет связать один объект (ключ), с другим объектом (значение). Естественно, ключ Map должен быть уникальным, и он используется для последующего извлечения значений. Коллекция Map Java может содержать только объекты и строго регламентирует их поведение.

Поскольку Map является интерфейсом, его экземпляр нельзя создать непосредственно, и необходимо использовать одну из реализаций интерфейса Map: HashMap, LinkedHashMap или TreeMap. HashMap обеспечивает максимальную скорость выборки, а порядок хранения его элементов не очевиден. TreeMap хранит ключи отсортированными по возрастанию, а LinkedHashMap хранит ключи в порядке вставки, но не обеспечивает скорость поиска HashMap.

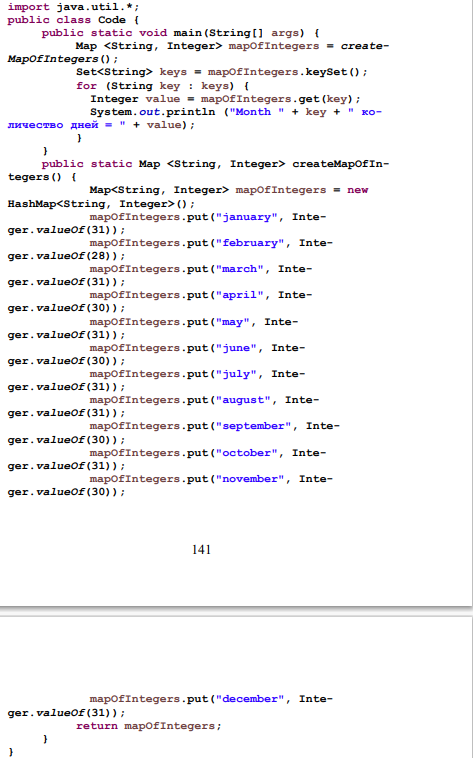
Ниже некоторые методы, которые можно выполнять с Map:

* put (String key, value) - поместить объект в Map;
* variable get(String key) - извлечь объект из Map;
* int size() - возвращает количество пар "ключ-значение" в отображении;
* Set keySet() - возвращает набор, содержащий ключи вызывающего отображения. Метод предоставляет ключи вызывающего отображения в виде набора и позволяет организовать перебор значений отображения;
* toString() - вывести содержимое в виде фигурных скобок, где ключи и значения разделяются знаком равенства. Ключи слева, значения справа;
* boolean containsKey(Object k) - возвращает значение true, если вызывающее отображение содержит ключ k, false - в противном случае;
* boolean containsValue(Object v) - возвращает значение true, если вызывающее отображение содержит значение v, false - в противном случае;
* boolean equals(Object o) - возвращает значение true, если параметр o - это отображение, содержащее одинаковые значения, false - в противном случае;
* V get(Object k) - возвращает значение, ассоциированное с ключом k. Возвращает значение null, если ключ не найден;
* boolean isEmpty() - возвращает значение true, если вызывающее отображение пустое, false - в противном случае;
* V put(K k, V v) - помещает элемент в вызывающее отображение, переписывая любое предшествующее значение, ассоциированное с ключом. Возвращает null, если 140 ключ ранее не существовал. В противном случае возвращается предыдущее значение, связанное с ключом;
* V remove(Object k) - удаляет элемент, ключ которого равен k;
* Collection values() - возвращает коллекцию, содержащую значения отображения.

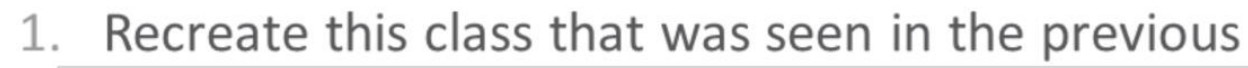
Использование Map

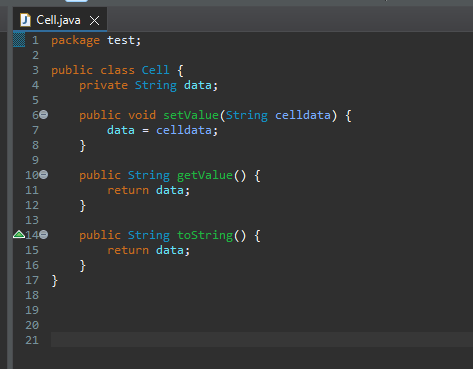


Использование Set совместно с Map



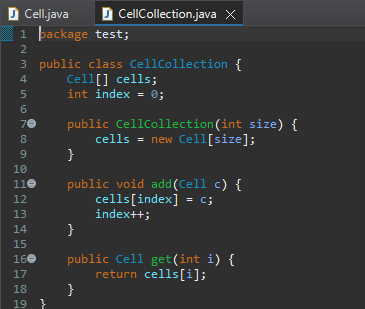
3-2 Collection





Изображение выглядит как текст, Шрифт, белый, алгебра

Контент, сгенерированный ИИ, может содержать ошибки.



Изображение выглядит как текст, Шрифт, белый, чек

Контент, сгенерированный ИИ, может содержать ошибки.

1. **Каково назначение полей экземпляра?**

В классе Cell:

* + data: строка (String), хранящая значение ячейки.

В классе CellCollection:

* + cells: массив объектов Cell, который используется для хранения нескольких ячеек.
  + index: целочисленное поле, отслеживающее позицию, куда будет добавлен следующий объект Cell.

1. **Что делает конструктор?**

В классе CellCollection конструктор создает массив cells заданного размера, резервируя память для хранения объектов Cell.

1. **Что делает метод add?**

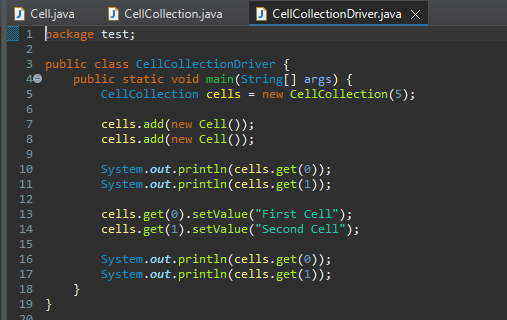
Метод add в CellCollection добавляет объект Cell в массив cells в позицию, указанную index, затем увеличивает index на 1, чтобы следующая ячейка добавлялась в следующую позицию массива.

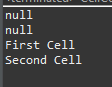
1. **Что делает метод get?**

Метод get в CellCollection возвращает объект Cell из массива cells по заданному индексу i.

**Задание 2**







**Задание 3**

**Изображение выглядит как текст, Шрифт, белый, алгебра

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, Шрифт, снимок экрана, документ

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.**

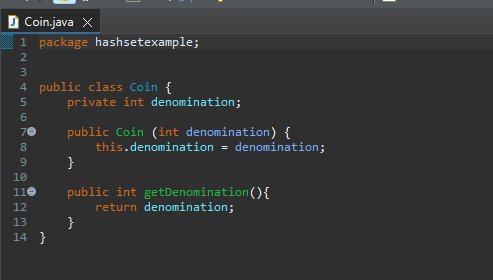
**Изображение выглядит как текст, Шрифт, снимок экрана, типография

Контент, сгенерированный ИИ, может содержать ошибки.**

**Задание 4**

**Изображение выглядит как текст, Шрифт, белый, типография

Контент, сгенерированный ИИ, может содержать ошибки.**

****

**Изображение выглядит как текст, чек, Шрифт, белый

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, Шрифт, белый, алгебра

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, электроника, снимок экрана, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, снимок экрана, Шрифт, алгебра

Контент, сгенерированный ИИ, может содержать ошибки.**

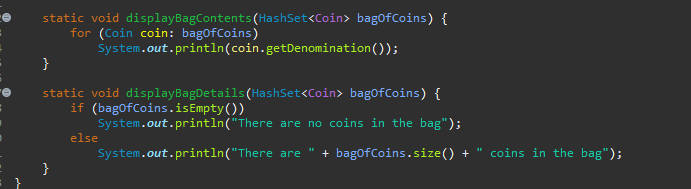
**Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, Шрифт, снимок экрана, документ

Контент, сгенерированный ИИ, может содержать ошибки.**

****

****

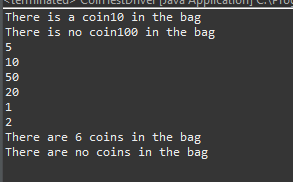
**Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, Шрифт, белый, чек

Контент, сгенерированный ИИ, может содержать ошибки.**

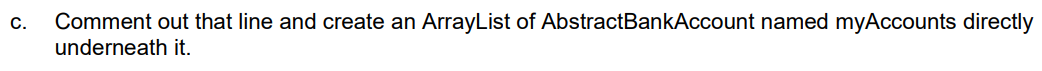
****

****

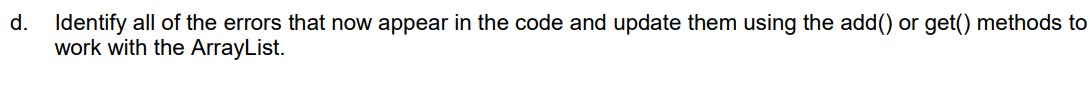
# Практика 3-2 Collections

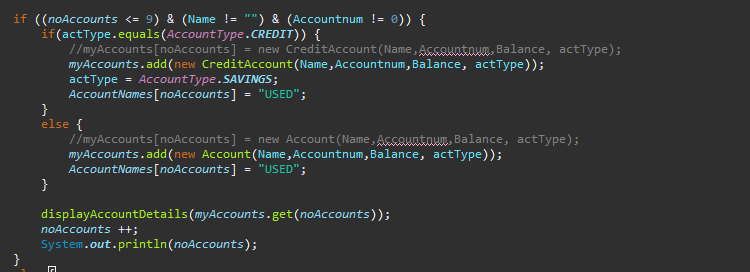
Изображение выглядит как текст, снимок экрана, Шрифт, линия

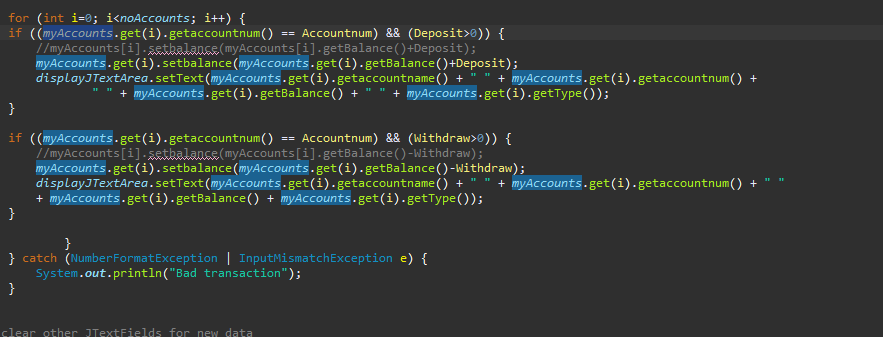
Контент, сгенерированный ИИ, может содержать ошибки.

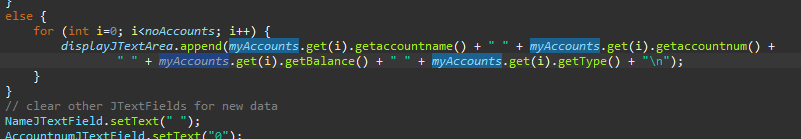






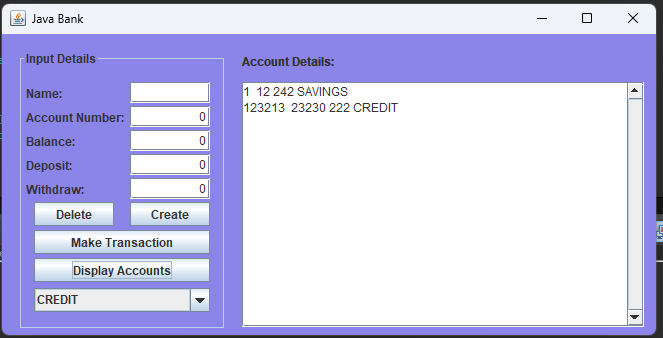












Изображение выглядит как текст, Шрифт, снимок экрана, число

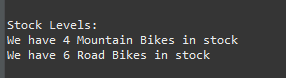
Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.



3. What is the difference between a set and a list?

1. Порядок элементов:
   * Список (List): Сохраняет порядок добавления элементов. Элементы в списке хранятся в том порядке, в котором они были добавлены, и их можно получить по индексу.
   * Множество (Set): Не сохраняет порядок элементов. Элементы в множестве не упорядочены, и доступ к ним осуществляется не по индексу, а с помощью итерации.
2. Дубликаты:
   * Список (List): Разрешает дубликаты. В списке могут быть несколько одинаковых элементов.
   * Множество (Set): Не разрешает дубликаты. Множество автоматически удаляет повторяющиеся элементы, гарантируя, что каждый элемент появляется только один раз.
3. Производительность:
   * Список (List): Операции добавления, удаления или поиска элемента могут занимать больше времени, особенно для больших списков, так как могут потребоваться сдвиги элементов или последовательный поиск.
   * Множество (Set): Как правило, обеспечивает более быстрое выполнение операций добавления, удаления и поиска (средняя сложность O(1)) благодаря использованию хеширования (в случае HashSet) или деревьев (в случае TreeSet).
4. Доступ к элементам:
   * Список (List): Элементы можно получить по индексу (например, list.get(2)), что позволяет произвольно обращаться к элементам по их позиции.
   * Множество (Set): Нельзя получить элементы по индексу. Вместо этого элементы можно только итерировать, но порядок их извлечения не гарантирован.

4. You decide you want to roll 2 dice and see what the frequency is of each possible number combination. Would you use a Set collection to do this? State your reason(s).

Нет, для этой задачи не стоит использовать коллекцию типа **Set**. Причины следующие:

1. **Отсутствие дубликатов**: Коллекция **Set** не разрешает дублирующиеся элементы. Однако при подбрасывании двух кубиков возможны одинаковые комбинации (например, выпадение 2 и 4 в одном броске, а затем 4 и 2 в другом), и вам нужно будет учитывать все эти случаи. **Set** автоматически удалит такие дубликаты, что сделает невозможным корректный подсчёт всех комбинаций.
2. **Неупорядоченность**: Множество **Set** не сохраняет порядок элементов. Для задачи подсчёта частоты каждой комбинации важно иметь возможность отслеживать, сколько раз каждая комбинация повторяется, и для этого полезно использовать структуру данных, которая позволяет хранить такие элементы в порядке их появления.

5. Using a collection create a variable that will store a list of countries (Strings). Your collection should not store duplicates, and order is not important. Test your code by adding 6 countries, one of which is a duplicate.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.



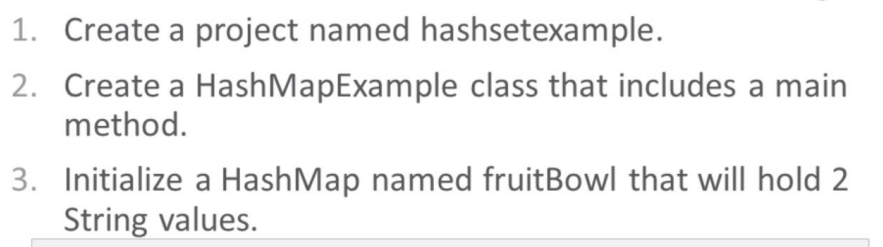
6. Would the following Collection.sort() statements both work? Explain your answer. HashSet countriesSet = new HashSet(); Collections.sort(countriesSet); ArrayList countriesList = new ArrayList(); Collections.sort(countriesList);

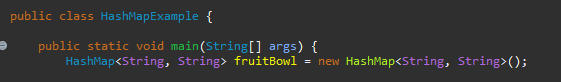
Не сработает.

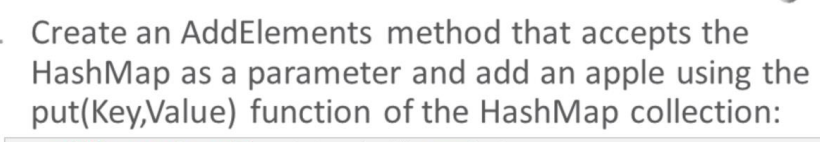
Причина: Метод Collections.sort() работает только с коллекциями, которые реализуют интерфейс List. Однако HashSet реализует интерфейс Set, а не List, и не имеет определенного порядка элементов. Set не гарантирует порядок элементов, и HashSet не поддерживает индексацию, необходимую для сортировки с помощью Collections.sort(). Поэтому попытка применить Collections.sort() к HashSet вызовет ошибку компиляции.

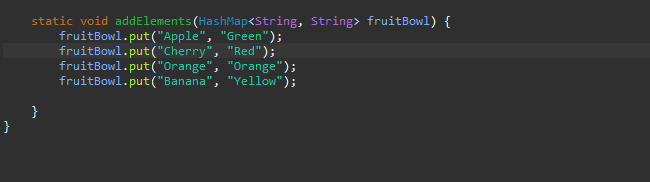
# 3-3 Collections

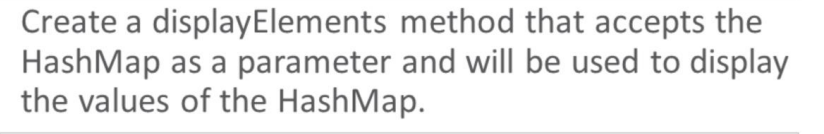
**Задание 1**

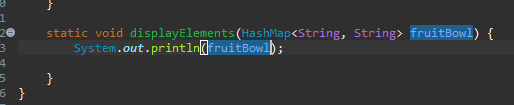
****



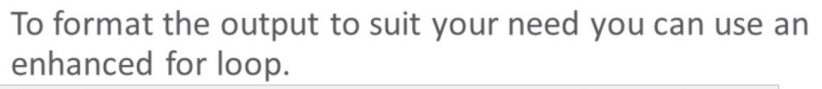


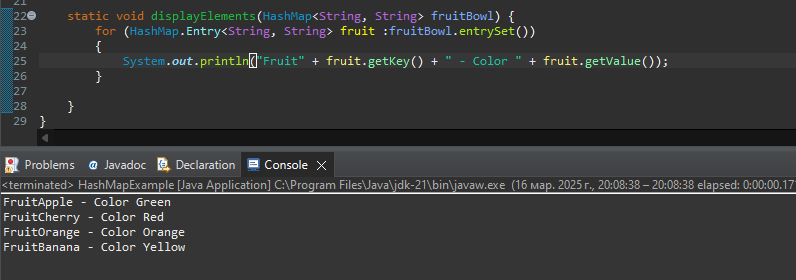


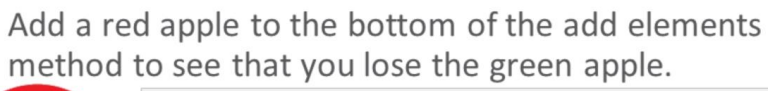


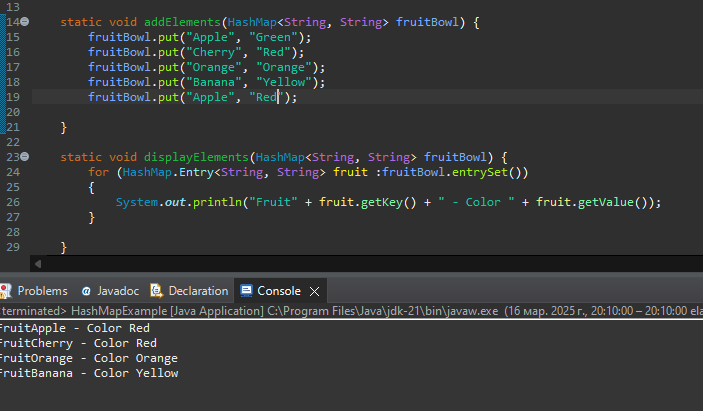
s

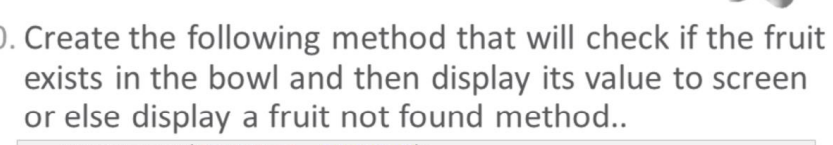




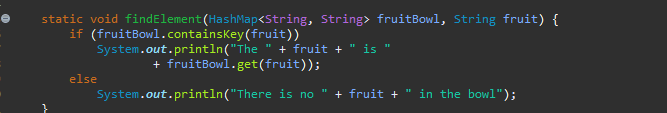


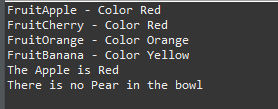












Задание 2

**Изображение выглядит как текст, Шрифт, чек, белый

Контент, сгенерированный ИИ, может содержать ошибки.**

****

****

**Изображение выглядит как текст, Шрифт, белый

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.**

**Задание 2**

**Изображение выглядит как текст, Шрифт, чек, белый

Контент, сгенерированный ИИ, может содержать ошибки.**

****

****

**Изображение выглядит как текст, Шрифт, белый, алгебра

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

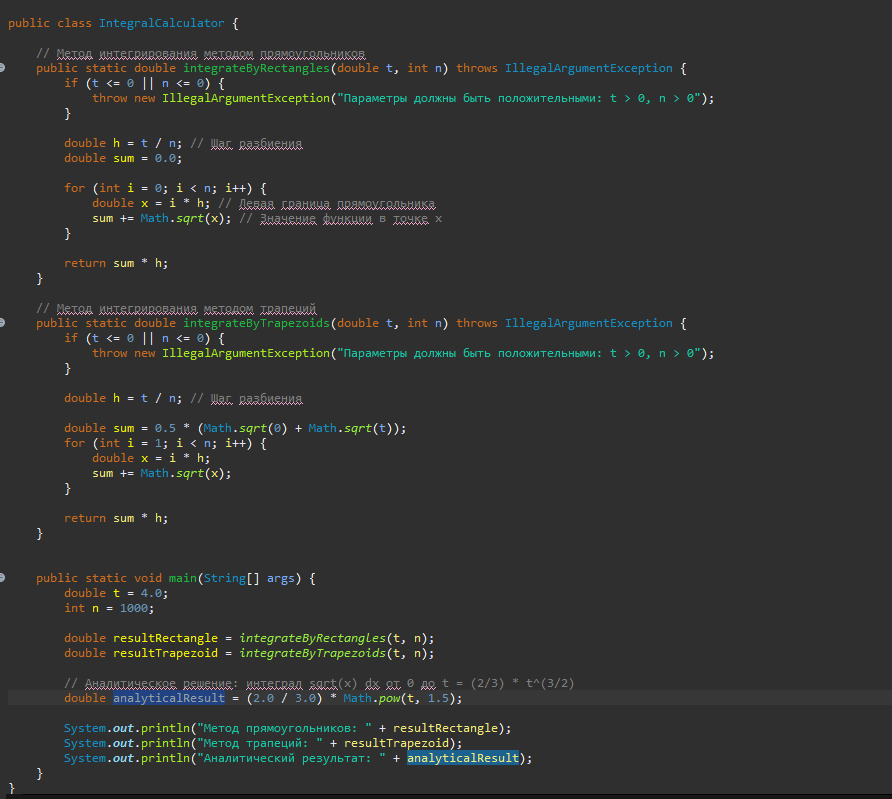
Контент, сгенерированный ИИ, может содержать ошибки.**

# IntegralCalculator

**Задание.**

Напишите на Java два метода, один из которых считает интеграл функции "корень квадратный из x" в диапазоне от 0 до t (где положительное вещественное t передатся в качестве параметра) методом прямоугольников, второй - методом трапеций. Число прямоугольников или трапеций задаете вторым целочисленным параметром. Оба метода возвращают вещественное значение. Контролируйте корректность значения входных параметров, при некорректном значении ваш метод должен выбрасывать исключение. В методе main вызовите методы с разными значениями параметров и напечатайте в консоль результаты. Сравните с результатом, полученным аналитически.

**Решение.**





**Стивенс Р. Алгоритмы. Теория и практическое применение. - М:Издательство "Э", 2016**

**ЧИСЛЕННЫЕ АЛГОРИТМЫ**

Численные алгоритмы используются для работы с числами. Они располагают величины в случайном порядке, разбивают числа на простые множители, находят наибольший общий делитель и определяют геометрические площади. И хотя такие алгоритмы применимы лишь в некоторых случаях, лежащие в их основе технологии очень полезны, например адаптивные алгоритмы, моделирование по методу Монте-Карло или использование таблиц для хранения промежуточных результатов.

**Рандомизация данных**

Рандомизация играет важную роль во многих приложениях. Она позволяет программе моделировать некоторые процессы, проверять поведение алгоритмов при работе со случайными входными величинами, а также искать ответы для сложных задач. Интеграция Монте-Карло, описанная в разделе «Численное интегрирование» текущей главы, использует случайно выбранные точки для оценки площади сложной геометрической фигуры. Первый шаг в любом алгоритме рандомизации — генерирование случайных чисел.

**Генерирование случайных величин**

В действительности ни один алгоритм, используемый компьютером для получения чисел, не является «случайным». Чтобы добиться абсолютной непредсказуемости, необходимо использовать нечто отличное от компьютерной программы. Одним из таких вариантов мог бы служить детектор излучения, измеряющий частицы, которые испускает радиоактивный образец. Примером простого и общего метода создания псевдослучайных чисел является линейный конгруэнтный генератор.

**Обеспечение равноправия**

Обычно в программах нужно использовать равноправные ГПСЧ. Равноправным ГПСЧ называется тот, который производит все возможные выходные данные с одинаковой вероятностью. ГПСЧ, являющийся неравноправным, называют неправильным. Например, случай с монетой, падающей орлом в 2/3 случаев, является неправильным.

Многие языки программирования располагают методами, производящими случайные числа в любых желаемых пределах, но если требуется написать код для преобразования величин ГПСЧ в особый диапазон, нужно быть осторожным при соблюдении равноправия.

**Получение равноправия от неправильных источников**

Даже если ГПСЧ является неправильным, есть способ генерирования равноправных чисел. Предположим, вы считаете монету неправильной. Вам сложно спрогнозировать вероятность того, выпадет у вас орел или решка, однако можно предположить, что эта вероятность не равна 0,5.

**Рандомизация массивов**

Довольно распространенной задачей в программах является рандомизация элементов массива. Допустим, программе календарного планирования нужно назначить смены для работников организации. Если программа организует смены в алфавитном порядке по мере их появления в базе данных или в каком-либо другом статистическом порядке, сотрудники, которых постоянно записывают на ночную смену, будут недовольны.

В некоторых алгоритмах также может использоваться случайность с целью предотвращения наихудшей ситуации. Например, стандартный алгоритм быстрой сортировки обычно работает хорошо, однако, если величины, которые он должен упорядочить, уже были отсортированы, производительность снижается. Одним из способов выхода из положения будет рандомизация величин перед их сортировкой.

**Генерирование неравномерных распределений**

В некоторых программах нужно генерировать псевдослучайные числа, распределяемые неравномерно. Нередко такие программы имитируют другие формы генерации случайных чисел. Например, программе может понадобиться сгенерировать числа между 2 и 12, чтобы сымитировать бросок двух шестигранных костей. Вы не можете просто выбрать псевдослучайные числа между 2 и 12, поскольку вероятность получения каждого числа при бросании двух костей отсутствует. Для решения задачи нужно имитировать бросание костей, чтобы сгенерировать два числа между 1 и 6, а затем сложить полученные результаты.

**Нахождение наибольшего общего делителя**

Наибольший общий делитель (НОД) двух целых чисел — наибольшее целое число, на которое они оба делятся без остатка. Например, НОД (60, 24) равен 12. Данная функция может показаться довольно странной, но на самом деле она очень полезна в криптографических задачах, которые широко используются в деловой сфере, в частности для защиты финансовых коммуникаций. В общем случае чтобы получить НОД, достаточно разложить два числа на множители и найти общие коэффициенты

**Возведение в степень**

Иногда для решения задачи требуется возвести число в целую степень. Если степень низкая, это не сложно. Например, 73 легко вычислить путем простого умножения 7 7 7 = 343. Для более высоких значений, таких как 7102 187 291, данный способ является очень медленным.

К счастью, есть способ ускорить процесс. Он основан на двух ключевых формулах:

Изображение выглядит как Шрифт, текст, белый, линия

Контент, сгенерированный ИИ, может содержать ошибки.

**Работа с простыми числами**

Как известно, простое число — это натуральное (целое положительное) число больше единицы, которое делится на единицу и на само себя. Все остальные натуральные числа больше единицы являются составными. Простые числа играют важную роль: они могут как облегчать, так и усложнять отдельные операции. Например, в некоторых видах криптографии, чтобы сделать алгоритм более надежным, используется произведение двух больших простых чисел.

**Нахождение простых множителей**

Самый легкий способ найти простые множители для числа — попытаться разделить его на все числа в промежутке между 2 и числом, которое меньше исходного значения на единицу. Если возможный множитель делит число нацело, сохраните его. Затем возьмите частное от деления и продолжите поиск с ним. Обратите внимание: перед тем как взять следующую цифру из ряда простых чисел, вы снова должны попробовать тот же самый множитель, поскольку исходное число может содержать более одного такого множителя.

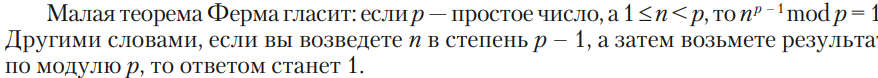
**Нахождение простых элементов**

Еще один метод нахождения простых чисел для указанного значения — решето Эратосфена. Он также хорош для достаточно малых величин, поскольку требует создания таблицы с записями для каждого анализируемого числа. Если значения будут слишком велики, понадобится неоправданно большой объем памяти. Идея заключается в том, чтобы создать таблицу с одной записью для каждого числа между 2 и указанным пределом. Для начала вычеркните все четные числа, не считая 2. Затем просмотрите таблицу, чтобы найти следующее невычеркнутое число (в данном случае это 3), и исключите все значения, кратные ему (не считая самого числа). Обратите внимание: некоторые значения уже могут быть вычеркнутыми, поскольку делились на 2. Повторите те же действия со следующим невычеркнутым числом и т. д., пока не дойдете до квадратного корня из верхнего предела. Все оставшиеся числа и будут искомыми простыми.

**Проверка на простоту**

Алгоритм, который раскладывает числа на множители можно использовать и для определения простых чисел. Идея заключается в следующем: если алгоритм не выполнит свою задачу, значит, число простое.

Как уже говорилось, производительность данной программы хороша для относительно малых чисел. Для числа, состоящего из 100 цифр, количество шагов алгоритма будет уже 50-значным. Даже самые быстрые компьютеры не способны выполнять такой объем операций в разумный промежуток времени. Но, к счастью, есть альтернатива, например тест простоты Ферма.



**Численное интегрирование**

Численное интегрирование (квадратура, или числовая квадратура) — использование численных методов для аппроксимации области под кривой, что определяется некоторой функцией. Чаще всего это функция одной переменной y = F(x), которая образует двумерную область. В некоторых случаях область может быть трехмерной и определяться функцией z = F(x, y). Допустимы и более многомерные варианты.

**Формула прямоугольников**

В этом способе для аппроксимации области под кривой используется несколько прямоугольников одинаковой ширины. Этот алгоритм просто делит область на прямоугольники постоянной ширины и высотой, равной величине функции с левого ребра прямоугольника. Затем он описывает петлю над прямоугольниками, добавляя их области.

**Формула трапеций**

Альтернативная стратегия для аппроксимации кривой — заменить прямоугольники трапециями. Единственная существенная разница между этим алгоритмом и тем, который работает с формулой прямоугольников, заключается в операторе, добавляющем область к каждой части. В данном случае используется формула для построения трапеций: область = ширина среднее длин параллельных сторон.

**Адаптивная квадратура**

Такая разновидность численной интеграции обнаруживает области, в которых аппроксимация имеет наибольшую погрешность, и совершенствует используемый для работы метод. Программа, использующая адаптивную квадратуру, ищет области, где фигуры не примыкают вплотную к кривой, и увеличивает их количество на этих участках.

**Интеграция Монте-Карло**

Это еще одна разновидность численной интеграции, при которой программа генерирует множество равномерно распределенных псевдослучайных точек, определяет, какие из них находятся внутри целевой области, а затем оценивает искомую площадь.

**Нахождение нулей**

Иногда программе нужно выяснить, где уравнение пересекает ось x. Другими словами, имея уравнение y = f(x), требуется найти x, при котором f(x) = 0. Такие величины называются корнями уравнения. Метод Ньютона (иногда его называют методом Ньютона — Рафсона) позволяет определить корни уравнения путем последовательной аппроксимации

**Резюме**

По сравнению с рандомизацией данных такие численные алгоритмы, как разложение на множители и нахождение наибольшего общего делителя, в чистом виде имеют довольно ограниченную сферу применения. Тем не менее используемые в них методы и идеи в отдельных ситуациях оказываются довольно ценными. Например, мысль о том, что алгоритм может быть вероятностным, пригодится для написания других алгоритмов, не работающих с отличной достоверностью (подобный вопрос может встретиться и на собеседовании).