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

**Tasks from presentation**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**3-1 Collection**

Graphical user interface, text, application, email

Description automatically generated

HashSet, List, Collection, ArrayList, Set

1. **Всё в JavaBank уже есть!**

2. **ВСЁ В BikeList.java**

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

List это упорядоченный список, которым можно управлять и к каждому элементу можно получить доступ по индексу.

Set это неупорядоченный список, который не содержит повторяющиеся элементы.

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).

Он не подойдет, потому что не может хранить повторящиеся элементы.

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.

**Text

Description automatically generated**

6. Would the following Collection.sort() statements both work? Explain your answer.

sort() нельзя применить к HashSet, так как это неупорядоченный список, его нельзя отсортировать;

вместе с ArrayList sort() сработает, так как элементы в нём хранятся последовательно.

**Дубаков В. В.**

**Введение в ООП на Java**

**Коллекции Java**

Язык Java имеет мощную структуру интерфейсов и классов в java.util, которые позволяет создавать и управлять группами объектов разного типа. Типичный класс коллекции реализует несколько интерфейсов, которые представляют собой хорошо разработанную иерархию. Например, ArrayList реализует интерфейс List, который расширяет Collection.

Collection является корневым интерфейсом в иерархии, из которого производятся интерфейсы List, Queue и Set.

Иерархия интерфейсов.

Diagram

Description automatically generated

В представленной иерархии показаны следующие интерфейсы коллекции:

* 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. Существуют многие другие дисциплины обслуживания. Здесь интерфейс Queue приводится для полноты, но не рассматривается в дальнейшем.

**List ‐ Список**

Список List представляет собой коллекцию, которая, по определению, упорядочена, то есть является последовательностью. Поскольку List упорядочен, можно полностью управлять местом размещения его элементов. Коллекция Java List может содержать только объекты и строго регламентирует их поведение. Используя коллекции, программист применяет существующие структуры данных, не заботясь о том, как они реализованы.

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

Используя List, обычно работают с его реализацией в ArrayList и описывают следующим образом:

List <Object> name=new ArrayList<Object>();

**Объявление формального типа**

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

**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**

На практике список содержит записи файлов или бизнес-объекты, и для их обработки может потребоваться полный перебор элементов. В этой связи важной особенностью коллекций в Java является наличие т.н. итераторов. Итератор (Iterator)— особый класс, который позволяет организовать перемещение по элементам коллекции и удалять её элементы в цикле. Все коллекции из java.util реализуют интерфейс Collection, который, в свою очередь, расширяет интерфейс java.lang.Iterable.

Применительно к конкретной имеющейся коллекции, например, List listOfIntegers = new ArrayList(); итератор может описан следующим образом: Iterator iterator = listOfIntegers.iterator();

while (iterator.hasNext()) {

 System.out.println(iterator.next());

 }

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().

**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.  
 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.

Отличие HashSet, LinkedHashSet или TreeSet в том, что первый хранит элементы в случайном порядке, второй в порядке записи, третий в порядке возрастания.

**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.

Интерфейс отображения описывается interface Map<K,V>. В параметре K указывается тип ключей, в V - тип хранимых значений.

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 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.

**Блинов, И.Н., Романчик, В. С.**

**Java. Методы программирования**

**Коллекции**

Коллекции — это хранилища или контейнеры, поддерживающие различные способы накопления и упорядочения объектов с целью обеспечения возможностей эффективного доступа к ним. Они представляют собой реализацию абстрактных структур данных, поддерживающих три основные операции:

• добавление нового элемента в коллекцию;

• удаление элемента из коллекции;

• изменение элемента в коллекции.

Интерфейсы коллекций:

Diagram

Description automatically generated

Map — карта отображения вида «ключ-значение»;

Collection — вершина иерархии коллекций List, Set;

List — специализирует коллекции для обработки списков;

Set — специализирует коллекции для обработки множеств, содержащих уникальные элементы.

Все классы коллекций реализуют также интерфейсы Serializable, Cloneable (кроме WeakHashMap). Кроме того, классы, реализующие интерфейсы List и Set, реализуют также интерфейс Iterable.

В интерфейсе Collection определены методы, которые работают на всех коллекциях:

boolean add(E obj) — добавляет obj к вызывающей коллекции и возвращает true, если объект добавлен, и false, если obj уже элемент коллекции;

boolean remove(Object obj) — удаляет obj из коллекции;

boolean addAll(Collection c) — добавляет все элементы коллекции к вызывающей коллекции;

void clear() — удаляет все элементы из коллекции;

boolean contains(Object obj) — возвращает true, если вызывающая коллекция содержит элемент obj;

boolean equals(Object obj) — возвращает true, если коллекции эквивалентны;

boolean isEmpty() — возвращает true, если коллекция пуста; Iterator iterator() — извлекает итератор;

int size() — возвращает количество элементов в коллекции;

Object[] toArray() — копирует элементы коллекции в массив объектов;

T[] toArray(T a[]) — копирует элементы коллекции в массив объектов определенного типа.

Интерфейс Iterator используется для построения объекта, который обеспечивает доступ к элементам коллекции. К этому типу относится объект, возвращаемый методом iterator(). Такой объект позволяет просматривать содержимое коллекции последовательно, элемент за элементом. Позиции итератора располагаются в коллекции между элементами. В коллекции, состоящей из N элементов, существует N+1 позиций итератора.

Методы интерфейса Iterator:

boolean hasNext() — проверяет наличие следующего элемента, а в случае его отсутствия (завершения коллекции) возвращает false. Итератор при этом остается неизменным;

E next() — возвращает ссылку на объект, на который указывает итератор, и передвигает текущий указатель на следующий, предоставляя доступ к следующему элементу. Если следующий элемент коллекции отсутствует, то метод next() генерирует исключение NoSuchElementException;

void remove() — удаляет объект, возвращенный последним вызовом метода next(). Если метод next() до вызова remove() не вызывался, то будет сгенерировано исключение IllegalStateException.

**Списки**

Класс ArrayList — динамический массив объектных ссылок. Реализует интерфейсы List, Collection, Iterable.

void add(int index, E element) — вставляет element в позицию, указанную в index;

E remove(int index) — удаляет объект из позиции index;

E set(int index, E element) — заменяет объект в позиции index, возвращает при этом удаляемый элемент;

void addAll(int index, Collection c) — вставляет в вызывающий список все элементы коллекции с, начиная с позиции index;

E get(int index) — возвращает элемент в виде объекта из позиции index;

int indexOf(Object ob) — возвращает индекс указанного объекта;

List subList(int fromIndex, int toIndex) — извлекает часть коллекции в указанных границах.

**Метасимвол в коллекциях**

Метасимволы используются при параметризации коллекций для расширения возможностей самой коллекции и обеспечения ее типобезопасности. Например, если параметр метода предыдущего примера изменить с List<Order> на List<? extends Order>, то в метод можно будет передавать коллекции, параметризованные любым допустимым типом, а именно классом Order и любым его подклассом, что невозможно при записи без анонимного символа.

Но в методе нельзя будет добавить к коллекции новый элемент, пусть даже и допустимого типа, так как компилятору неизвестен заранее тип параметризации списка.

**Интерфейс ListIterator**

Для доступа к элементам списка может также использоваться интерфейс ListIterator<E>, который позволяет получить доступ сразу в необходимую программисту позицию списка вызовом метода listIterator(int index). Интерфейс ListIterator<E> расширяет интерфейс Iterator<E> и предназначен для обработки списков и их вариаций. Наличие методов E previous(), int previousIndex() и boolean hasPrevious() обеспечивает обратную навигацию по списку. Метод int nextIndex() возвращает номер следующего итератора. Метод void add(E obj) позволяет вставлять элемент в список текущей позиции. Вызов метода void set(E obj) производит замену текущего элемента списка на объект, передаваемый методу в качестве параметра

**Интерфейс Comparator**

Реализация метода equals() класса Object предоставляет возможность проверить, эквивалентен один экземпляр другому или нет. На практике возникает необходимость сравнения объектов на больше/меньше либо равно.

При реализации интерфейса java.util.Comparator появляется возможность сортировки списка объектов конкретного типа по правилам, определенным для этого типа. Контракт интерфейса подразумевает реализацию метода int compare(T ob1, T ob2), принимающего в качестве параметров два объекта, для которых должно быть определено возвращаемое целое значение, знак которого и определяет правило сортировки. Метод public abstract boolean equals(Object obj), также объявленный в интерфейсе Comparator<T>, очень рекомендуется переопределять, если экземпляр класса будет использоваться для хранения информации. Это необходимо для исключения противоречивой ситуации, когда для двух объектов метод compare() возвращает 0, т. е. сообщает об их эквивалентности, в тоже время метод equals() для этих же объектов возвращает false, так как данный метод не был никем определен и была использована его версия из класса Object. Кроме того, наличие метода equals() обеспечивает корректную работу метода семантического поиска и проверки на идентичность contains(Object o), определенного в интерфейсе java.util.Collection, а следовательно, реализованного в любой коллекции.

Метод compare() автоматически вызывается при сортировке списка методом: static<T> void sort(List<T> list, Comparator<? super T> c) класса Collections, в качестве первого параметра принимающий коллекцию, в качестве второго — объект-comparator, из которого извлекается и применяется правило сортировки.

**Класс LinkedList и интерфейс Queue**

Коллекция LinkedList реализует связанный список.

Связанный список хранит ссылки на объекты отдельно вместе со ссылками на следующее и предыдущее звенья последовательности, поэтому часто называется двунаправленным списком. Операции добавления и удаления выполняются достаточно быстро, в отличие от операций поиска и навигации. В этом классе объявлены методы, позволяющие манипулировать им как очередью, двунаправленной очередью и т. д. Двунаправленный список кроме обычного имеет особый «нисходящий» итератор, позволяющий двигаться от конца списка к началу, и извлекается методом descendingIterator().

Для манипуляций с первым и последним элементами списка в LinkedList<E> реализованы методы:

void addFirst(E ob), void addLast(E ob) — добавляющие элементы в начало и конец списка;

E getFirst(), E getLast() — извлекающие элементы;

E removeFirst(), E removeLast() — удаляющие и извлекающие элементы;

E removeLastOccurrence(E elem), E removeFirstOccurrence(E elem)  — удаляющие и извлекающие элемент, первый или последний раз встречаемый в списке

Методы интерфейса Queue:

boolean add(E o) — вставляет элемент в очередь, если же очередь полностью заполнена, то генерирует исключение IllegalStateException;

boolean offer(E o) — вставляет элемент в очередь, если возможно;

E element() — возвращает, но не удаляет головной элемент очереди;

E peek() — возвращает, но не удаляет головной элемент очереди, возвращает null, если очередь пуста;

E poll()  — возвращает и удаляет головной элемент очереди, возвращает null, если очередь пуста;

E remove() — возвращает и удаляет головной элемент очереди.

Методы element() и remove() отличаются от методов peek() и poll() тем, что генерируют исключение NoSuchElementException, если очередь пуста.

При всей схожести списков ArrayList и LinkedList существуют серьезные отличия, которые необходимо учитывать при использовании коллекций в конкретных задачах. Если необходимо осуществлять быструю навигацию по списку, то следует применять ArrayList, так как перебор элементов в LinkedList осуществляется на порядок медленнее. С другой стороны, если требуется часто добавлять и удалять элементы из списка, то уже класс LinkedList обеспечивает значительно более высокую скорость переиндексации.

**Интерфейс Deque и класс ArrayDeque**

Интерфейс Deque определяет «двунаправленную» очередь и, соответственно, методы доступа к первому и последнему элементам двусторонней очереди. Реализацию этого интерфейса можно использовать для моделирования стека. Методы обеспечивают удаление, вставку и обработку элементов. Каждый из эиих методов существует в двух формах. Одни методы создают исключительную ситуацию в случае неудачного завершения, другие возвращают какое-либо из значений (null или false в зависимости от типа операции). Вторая форма добавления элементов в очередь сделана специально для реализаций Deque, имеющих ограничение по размеру. В большинстве реализаций операции добавления заканчиваются успешно. Методы addFirst(), addLast() вставляют элементы в начало и в конец очереди соответственно. Метод add() унаследован от интерфейса Queue и абсолютно аналогичен методу addLast() интерфейса Deque. Объявить двуконечную очередь на основе связанного списка можно, например, следующим образом:

Deque dq = new LinkedList<>();

Класс ArrayDeque быстрее, чем Stack, если используется как стек, и быстрее, чем LinkedList, если используется в качестве очереди.

**Множества**

Интерфейс Set объявляет поведение коллекции, не допускающей дублирования элементов. Интерфейс SortedSet наследует Set и объявляет поведение набора, отсортированного в возрастающем порядке, заранее определенном для класса. Интерфейс NavigableSet существенно облегчает поиск элементов, например, расположенных рядом с заданным.

**Карты отображений**

Карта отображений — это объект, который хранит пару «ключ–значение». Поиск объекта (значения) облегчается по сравнению с множествами за счет того, что его можно найти по его уникальному ключу. Уникальность объектовключей должна обеспечиваться переопределением методов hashCode() и equals() или реализацией интерфейсов Comparable, Comparator пользовательским классом.

**Унаследованные коллекции**

В ряде распределенных приложений, например с использованием сервлетов, до сих пор применяются коллекции, более медленные в обработке, но при этом потокобезопасные (thread-safety), существовавшие в языке Java с момента его создания, а именно карта Hashtable<K,V>, список Vector<E> и перечисление (аналог итератора) Enumeration<E>. Все они также были параметризованы, но сохраняют возможность одновременного доступа из конкурирующих потоков.

**Алгоритмы класса Collections**

Класс java.util.Collections содержит большое количество статических методов, предназначенных для манипулирования коллекциями.

С применением предыдущих версий языка было разработано множество коллекций, в которых никаких проверок нет, следовательно, при их использовании нельзя гарантировать, что в коллекцию не будет помещен «посторонний» объект. Для этого в класс Collections был добавлен новый метод:

static <E> Collection checkedCollection(Collection<E> c, Class<E> type)

Этот метод создает коллекцию, проверяемую на этапе выполнения, т. е. в случае добавления «постороннего» объекта генерируется исключение ClassCastException.

**Глава 2.**

**Численные алгоритмы.**

**Генерирование случайных величин**

К сожалению, генераторы истинно случайных чисел являются довольно сложными и медленными, поэтому в большинстве задач используется более быстрый генератор псевдослучайных чисел (ГПСЧ). Если подобранные с его помощью величины в определенном смысле достаточно «случайны», на их основе программа демонстрирует хорошие результаты.

**Генерирование величин**

Примером простого и общего метода создания псевдослучайных чисел является линейный конгруэнтный генератор, использующий следующую зависимость для формирования величин:

****

Величина X0 называется начальным числом. Она инициализирует генератор, поэтому различные величины X0 дают неодинаковые комбинации чисел.

Поскольку все значения в числовой последовательности берутся по модулю M, как только генератор достигает максимума, он производит число, полученное им ранее, а последовательность чисел повторяется с того места.

**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.

**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.

В общем случае чтобы получить НОД, достаточно разложить два числа на множители и найти общие коэффициенты. Однако еще около 300 г. до н. э. греческий математик Евклид в своем трактате «Начала» описал более быстрый способ, который представлен в следующем псевдокоде. Поскольку алгоритм создан на основе работы Евклида, он называется евклидовым, или алгоритмом Евклида.

**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 текст, доска

Автоматически созданное описание

Первая позволяет быстро вычислить степень числа A, возводя в квадрат это же число в исходной степени; вторая помогает комбинировать степени любым удобным для вас образом.

**Нахождение простых множителей**

Следует проверять множители только до квадратного корня числа. Если n = p q, то p или q должно быть меньше либо равно sqrt (n). (Если p и q больше sqrt (n), их произведение превысит n.) Проверив возможные множители до sqrt (n), вы найдете наименьший среди них, а, поделив n на такой множитель, определите еще один. Это сократит время работы до O(sqrt (n)).

**Нахождение простых элементов**

Предположим, вашей программе нужно выбрать большое простое число (подобные задания не редкость в алгоритмах шифрования). Один из способов сделать это — использовать код, описанный ранее, и проверить, являются ли числа простыми. С разумно малыми значениями такой вариант сработает хорошо, но для очень больших чисел он может оказаться чересчур медленным.

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.

**Проверка на простоту**

Малая теорема Ферма гласит: если p — простое число, а 1 <= n < p, то n p – 1 mod p = 1. Другими словами, если вы возведете n в степень p – 1, а затем возьмете результат по модулю p, то ответом станет 1.