Лекция 5

Коллекции: LinkedList, HashSet, HashMap. Date - дата.

Коллекции

- Коллекциями/контейнерами в Java принято называть классы, основная цель которых – хранить набор других элементов. Пример такого класса, с которым вы уже знакомы, есть – это ArrayList.

- В Java коллекции делятся на три основных группы: Set — множество, List — список, Map — словарь (или карта).

- Все коллекции и контейнеры сразу после создания ничего в себе не хранят. Но в них можно постепенно добавлять элементы – тогда они будут динамически изменять свой размер.

Set

- Представьте много туфель сброшенных в кучу — это Set — множество. В set можно добавить элемент, поискать его или удалить. Но у элементов там нет строго заданного порядка!

List

- А теперь представьте ту же кучу обуви, но выставленную вдоль стенки. Теперь в ней появился порядок. У каждого элемента есть его номер. И можно просто взять «пару номер 7» по её номеру. Это — List — список. Можно добавить элемент в начало списка или в середину, или убрать его, и сделать это просто по его номеру.

Map

- Представьте ту же обувь, но теперь у каждой пары есть табличка с именем «Коля», «Вася», «Аня». Это Map — мэп. У каждого элемента есть его уникальное название, по которому к нему можно было обращаться. Уникальное название элемента ещё называют «ключом». А Map — это набор пар «ключ»-«значение». Ключ не обязательно должен быть строкой, он может быть любым типом. **Map**, у которого тип ключа — **Integer**, — это фактически **List** (с некоторыми отличиями).

Различные реализации интерфейсов List, Set и Map.

Интерфейс	Класс/Реализация	Описание
List	ArrayList	Список
	LinkedList	Список
	Vector	Вектор
	Stack	Стек
Set	HashSet	Множество
	TreeSet	Множество
	SortedSet	Отсортированное множество
Map	HashMap	Карта/Словарь
	TreeMap	Карта/Словарь
	SortedMap	Отсортированный словарь
	Hashtable	Хеш-таблица

- У элементов списка (List) есть четко заданный порядок, поэтому их можно вывести просто по номеру. У Set и Map строго заданного порядка элементов нет. Собственно говоря, порядок их элементов может меняться при удалении или добавлении какого-нибудь элемента.

- Поэтому для работы с элементами коллекций были придуманы специальные объекты — **итераторы**. С их помощью можно пройтись по всем элементам коллекции, даже если у них нет номера, а только имена (Map), или вообще нет имён — Set.

Вывод на экран элементов Set

```
public static void main(String[] args)
{
   Set<String> set = new HashSet<String>();
   set.add("Mama");
   set.add("Mila");
   set.add("Ramu");

   //получение итератора для множества
   Iterator<String> iterator = set.iterator();

   while (iterator.hasNext())       //проверка, есть ли ещё элементы
   {
      //получение текущего элемента и переход на следующий
      String text = iterator.next();

      System.out.println(text);
   }
}
```

Вывод на экран элементов List

```
public static void main(String[] args)
{
   List<String> list = new ArrayList<String>();
   list.add("Mama");
   list.add("Mila");
   list.add("Ramu");

   Iterator<String> iterator = list.iterator();//получение итератора для списка

   while (iterator.hasNext())     //проверка, есть ли ещё элементы
   {
      //получение текущего элемента и переход на следующий
      String text = iterator.next();

      System.out.println(text);
   }
}
```

Вывод на экран элементов Map

```
public static void main(String[] args)
{
   //все элементы хранятся в парах
   Map<String, String> map = new HashMap<String, String>();
   map.put("first", "Mama");
   map.put("second", "Mila");
   map.put("third", "Ramu");

   Iterator<Map.Entry<String, String>> iterator = map.entrySet().iterator();

  while (iterator.hasNext())
   {
      //получение «пары» элементов
      Map.Entry<String, String> pair = iterator.next();
      String key = pair.getKey();          //ключ
      String value = pair.getValue();      //значение
      System.out.println(key + ":" + value);
   }
}
```

- Сначала мы получаем у коллекции специальный объект-iterator. У него есть всего два метода
  1. Метод next() возвращает очередной элемент коллекции.
  2. Метод hasNext() проверяет, есть ли еще элементы, которые не возвращал next().

- Сначала надо вызвать у коллекции метод iterator(), чтобы получить объект-итератор.

- Затем в цикле, пока есть еще неполученные элементы, получаем их по одному. Получаем очередной элемент коллекции с помощью вызова next(), а проверяем, есть ли еще элементы в итераторе с помощью hasNext().

- В Java есть сокращённая запись работы с итераторами. По аналогии с **while**, в **for** был добавлен еще один специальный оператор «**for each**» — «**для каждого**». Обозначается тоже ключевым словом **for**.

- Оператор for each используется только при работе с коллекциями и контейнерами. В нем неявно используется итератор, но мы видим уже полученный элемент.

Длинная запись	Короткая запись

public static void main(String[] args)
{
Set<String> set = **new HashSet**<String>();
set.add("Mama");
set.add("Mila");
set.add("Ramu");
 	
Iterator<String> iterator = set.iterator();
while (iterator.hasNext())
{
String text = iterator.next();
System.out.println(text);
}
}
	public static void main(String[] args)
	{
	Set<String> set = **new HashSet**<String>();
	set.add("Mama");
	set.add("Mila");
	set.add("Ramu");
	
	for (String text : set)
	{
	System.out.println(text);
	}
	}

в правой нижней таблице нет ни зелёных, ни красных слов. Фактически 3 строки заменяются на одну:	
Iterator<String> iterator = set.iterator(); **while** (iterator.hasNext()) { String text = iterator.next();	**for** (String text : set)

Вывод на экран элементов Set

```
public static void main(String[] args)
{
   Set<String> set = new HashSet<String>();
   set.add("Mama");
   set.add("Mila");
   set.add("Ramu");

   for (String text : set)
   {
      System.out.println(text);
   }
}
```

Вывод на экран элементов List

```
public static void main(String[] args)
{
   List<String> list = new ArrayList<String>();
   list.add("Mama");
   list.add("Mila");
   list.add("Ramu");

   for (String text : list)
   {
      System.out.println(text);
   }
}
```

Вывод на экран элементов Map

```
public static void main(String[] args)
{
   Map<String, String> map = new HashMap<String, String>();
   map.put("first", "Mama");
   map.put("second", "Mila");
   map.put("third", "Ramu");

   for (String key : map.keySet()) //ключи
   {
      String value = map.get(key);//значение
      System.out.println(key + ":" + value);
   }
}
```

Класс Date

- С помощью этого класса можно хранить дату и время, а также измерять временные промежутки.

- Каждый объект типа Date хранит внутри себя время. А время хранится в очень интересном виде — количество миллисекунд, которые прошли с 1 января 1970 года, по Гринвичу.

- Это число настолько большое, что не влезает в **int**, приходится хранить его в **long**. Зато очень удобно считать разницу между двумя датами: отнял два числа и уже известна разница с точностью до миллисекунд. И к тому же устраняется проблема смены дат и перевода часов.

- А что самое интересное, каждый объект при создании инициализируется текущим временем. Чтобы узнать текущее время — достаточно просто создать объект.

Получение текущей даты:

```
public static void main(String[] args) throws Exception
{
   Date today = new Date();
   System.out.println("Current date: " + today);
}
```

Вычисление разницы между двумя датами:

```
public static void main(String[] args) throws Exception
{
   Date currentTime = new Date();          //получаем текущую дату и время
   Thread.sleep(3000);                  //ждём 3 секунды – 3000 миллисекунд
   Date newTime = new Date();             //получаем новое текущее время

   long msDelay = newTime.getTime() - currentTime.getTime(); //вычисляем разницу
   System.out.println("Time distance is: " + msDelay + " in ms");
}
```

Наступило ли уже некоторое время:

```
public static void main(String[] args) throws Exception
{
   Date startTime = new Date();

   long endTime = startTime.getTime() + 5000;   //    +5 секунд
   Date endDate = new Date(endTime);

   Thread.sleep(3000);           // ждем 3 секунды

   Date currentTime = new Date();
   if(currentTime.after(endDate))//проверяем что время currentTime после endDate
   {
      System.out.println("End time!");
   }
}
```

Сколько прошло времени с начала сегодняшнего дня:

```
public static void main(String[] args) throws Exception
{
   Date currentTime = new Date();
   int hours = currentTime.getHours();
   int mins = currentTime.getMinutes();
   int secs = currentTime.getSeconds();

   System.out.println("Time from midnight " + hours + ":" + mins + ":" + secs);
}
```

Сколько дней прошло с начала года:

```
public static void main(String[] args) throws Exception
{
   Date yearStartTime = new Date();
   yearStartTime.setHours(0);
   yearStartTime.setMinutes(0);
   yearStartTime.setSeconds(0);

   yearStartTime.setDate(1);     // первое число
   yearStartTime.setMonth(0);    // месяц январь, нумерация для месяцев 0-11

   Date currentTime = new Date();
   long msTimeDistance = currentTime.getTime() - yearStartTime.getTime();
   long msDay = 24 * 60 * 60 * 1000;  //сколько миллисекунд в одних сутках

   int dayCount = (int) (msTimeDistance/msDay); //количество целых дней
   System.out.println("Days from start of year: " + dayCount);
}
```

- Метод getTime() возвращает количество миллисекунд, которое хранится в объекте Date.

- Метод after() проверяет, что дата, у которой мы вызвали метод, идет после переданной в метод даты **after**.

- Методы getHours(), getMinutes(), getSeconds() возвращают количество часов, минут и секунд у объекта, у которого их вызвали.

- Более того, в последнем примере видно, что можно повлиять на дату/время, сохраненное в объекте **Date**. Мы получаем текущее время и дату, а затем сбрасываем часы, минуты и секунды в 0. Так же устанавливаем месяц в Январь, а день месяца в 1. Теперь объект yearStartTime хранит дату и время 1 января текущего года 0 часов 0 минут 0 секунд.
Потом снова получаем текущую дату currentTime, и вычисляем разницу в миллисекундах между двумя датами - msTimeDistance.
Затем делим msTimeDistance на количество миллисекунд в сутках и получаем полное количество дней, прошедшее с начала года до сегодняшнего дня!

ArrayList vs. LinkedList

- В таблице контейнеров и коллекций вы ранее видели, что у одного и того же интерфейса может быть несколько реализаций. Сейчас разберемся, в чем отличие **ArrayList** от **LinkedList**.

- Коллекции могут быть реализованы разными способами и нет единственного – самого правильного. При одном подходе одни операции являются быстрыми, а остальные медленными, при другом – все наоборот. Нет одного идеального, подходящего всем решения.

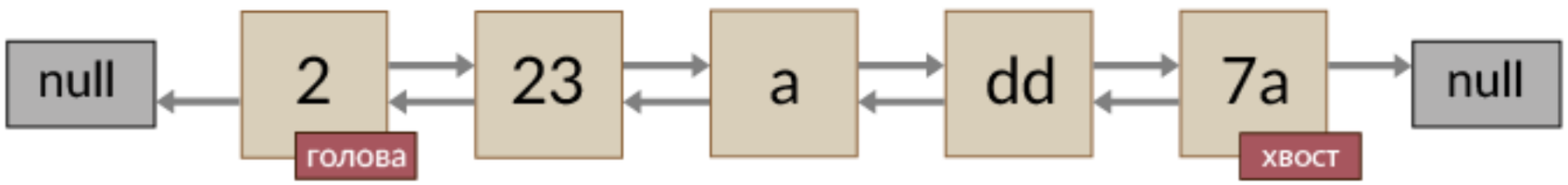
- Поэтому было решено сделать несколько реализаций одной и той же коллекции. И каждая реализация была оптимизирована для какого-то узкого набора операций. Так появились разные коллекции. Давай рассмотрим это на примере двух классов – **ArrayList** и **LinkedList**.

- **ArrayList** реализован внутри в виде обычного массива. Поэтому при вставке элемента в середину, приходится сначала сдвигать на один все элементы после него, а уже затем в освободившееся место вставлять новый элемент. Зато в нем быстро реализованы взятие и изменение элемента — операции get, set, так как в них мы просто обращаемся к соответствующему элементу массива.

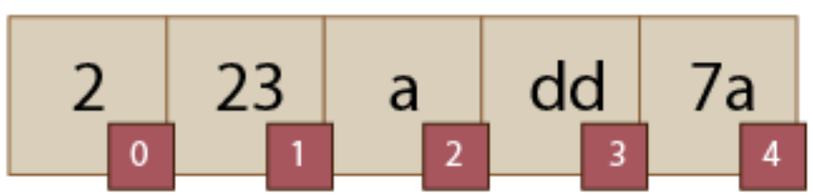
- **LinkedList** реализован внутри по-другому. Он реализован в виде связного списка: набора отдельных элементов, каждый из которых хранит ссылку на следующий и предыдущий элементы. Чтобы вставить элемент в середину такого списка, достаточно поменять ссылки его будущих соседей. А вот чтобы получить элемент с номером 130, нужно пройтись последовательно по всем объектам от 0 до 130. Другими словами операции set и get тут реализованы очень медленно.

ArrayList vs. LinkedList

Связный список (LinkedList)



Массив (Array и ArrayList)



	Описание	Операция	ArrayList	LinkedList
1	Взятие элемента	get	Быстро	Медленно
2	Присваивание элемента	set	Быстро	Медленно
3	Добавление элемента	add	Быстро	Быстро
4	Вставка элемента	add(i, value)	Медленно	Быстро
5	Удаление элемента	remove	Медленно	Быстро

- Если вы собираетесь вставлять (или удалять) в середину коллекции много элементов, то лучше использовать **LinkedList**. Во всех остальных случаях – **ArrayList**.

Set и Map

- **Set** – это множество – куча ненумерованных объектов. Главная особенность Set – в нем только уникальные объекты, т.е. они все разные. Вот что с ним можно делать:

	Операция	Метод
1	Добавлять элемент(ы)	add(), addAll()
2	Удалять элемент(ы)	remove(), removeAll()
3	Проверять, есть ли элемент(ы)	contains(), containsAll()

- **Map** – это множество пар. То же множество, но не одиноких элементов, а пар: ключ-значение. Единственное ограничение: первый объект в паре, называемый ключом, должен быть уникальным. В Map не может содержаться две пары с одинаковыми ключами.

	Операция	Метод
1	Получить множество всех пар	entrySet()
2	Получить множество всех ключей	keySet()
3	Получить множество всех значений	values()
4	Добавить пару	put(key, value)
5	Получить значение по ключу	get(key)
6	Проверить наличие «ключа»	containsKey(key)
7	Проверить наличие «значения»	containsValue(value)
8	Проверить что Map — пустой	isEmpty()
9	Очистить Map	clear()
10	Удалить элемент по ключу	remove(key)

Задачи

1. Нужно создать два списка - **LinkedList** и **ArrayList**.

```
public class Solution {
   public static Object createArrayList() {
      //напишите тут ваш код
   }

   public static Object createLinkedList() {
      //напишите тут ваш код
   }

   public static void main(String[] args) {
   }
}
```

2. Создать множество чисел(**Set**<Integer>), занести туда **20** различных чисел. Удалить из множества все числа больше **10**.

```
public class Solution {
   public static Set<Integer> createSet() {
      // напишите тут ваш код
   }

   public static Set<Integer> removeAllNumbersGreaterThan10(Set<Integer> set) {
      // напишите тут ваш код
   }

   public static void main(String[] args) {

   }
}
```

3. Есть класс **Cat** с полем имя (name, String).
Создать коллекцию **Map**<String, Cat> (реализация HashMap). Добавить в коллекцию **10 котов**, в качестве ключа использовать имя кота. Вывести результат на экран, каждый элемент с новой строки.

```
public class Solution {
   public static void main(String[] args) throws Exception {
      String[] cats = new String[]{"васька", "мурка", "дымка", "рыжик", "серый", "снежок", "босс", "борис", "визя",
"гарфи"};
      Map<String, Cat> map = addCatsToMap(cats);
      for (Map.Entry<String, Cat> pair : map.entrySet()) {
         System.out.println(pair.getKey() + " - " + pair.getValue());
      }
   }

   public static Map<String, Cat> addCatsToMap(String[] cats) {
      //напишите тут ваш код
   }

   public static class Cat {
      String name;

      public Cat(String name) {
         this.name = name;
      }

      @Override
      public String toString() {
         return name != null ? name.toUpperCase() : null;
      }
   }
}
```

4. Есть коллекция **Map**<String, Object> (реализация HashMap), туда занесли **10 различных пар объектов**. Вывести содержимое коллекции на экран, каждый элемент с новой строки.
   Пример вывода (тут показана только одна строка):
   Sim - 5

```
public class Solution {
    public static void main(String[] args) {
        Map<String, Object> map = new HashMap<>();
        map.put("Sim", 5);
        map.put("Tom", 5.5);
        map.put("Arbus", false);
        map.put("Baby", null);
        map.put("Cat", "Cat");
        map.put("Eat", new Long(56));
        map.put("Food", new Character('3'));
        map.put("Gevey", '6');
        map.put("Hugs", 111111111111L);
        map.put("Comp", (double) 123);

        //напишите тут ваш код
    }
}
```