Тема 4

Универсальные типы. Коллекции. Потоки

Макейчик Е.Г.

Обобщенное программирование

Обобщенное программирование – разновидность подхода к написанию программ, которая позволяет описать работу с различными типами данных некоторым единственным способом.

В C++ такой подход основан на использовании шаблонов (templates).

В Java его называют Generics. Можно создавать обобщения для классов и для методов.

Обобщения позволяют конструировать контейнеры для хранения данных различных типов. На них основан механизм коллекций.

Обобщенное программирование

```java
public class SimpleGeneric<T> {
    private T element;
    public T getElement() {
        return element;
    }
    public void setElement(T element) {
        this.element = element;
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        SimpleGeneric<String> sg1 = new SimpleGeneric<>();
        sg1.setElement("12345");

        SimpleGeneric<Integer> sg2 = new SimpleGeneric<>();
        sg2.setElement(99);
    }
}
```

Обобщенное программирование

В качестве параметров T могут выступать ссылочные типы (пользовательские классы и классы оболочки). Использовать примитивные типы нельзя. Можно создавать обобщения для методов.

```
public class GenMethod {
    public static <T> boolean eq(T one, T two) {
        return one.equals(two);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        System.out.println(GenMethod.eq( one: 77, two: 77));
    }
}
```

Обобщенное программирование

Метод для обработки массива

```
public class GenMethod {
    public static <T> boolean present(T[] arr, T item) {
        for (int i = 0; i < arr.length; i++)
            if (arr[i].equals(item))
                return true;
        return false;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Integer[] arr = new Integer[] {1,2,3,4};
        System.out.println(GenMethod.present(arr, item: 4));
    }
}
```

Обобщенное программирование

При создании обобщения можно использовать несколько параметров

```
public class Pair<T1, T2> {
    T1 object1;
    T2 object2;

    Pair(T1 one, T2 two) {
        object1 = one;
        object2 = two;
    }

    public T1 getFirst() {
        return object1;
    }
    public T2 getSecond() {
        return object2;
    }
}
```

```
Pair<Integer, String> pair = new Pair<Integer, String>( one: 18, two: "Пукач М.А.");
```

Обобщенное программирование

В современных версиях языка можно опускать указания типов в правой части выражения, например

```
public class Pair<T1, T2> {
    T1 object1;
    T2 object2;

    Pair(T1 one, T2 two) {
        object1 = one;
        object2 = two;
    }

    public T1 getFirst() {
        return object1;
    }
    public T2 getSecond() {
        return object2;
    }
}
```

```
Pair<Integer, String> pair = new Pair<>( one: 18, two: "Пукач М.А.");
```

Wildcards

В качестве параметров для шаблонов могут выступать подстановочные символы (wildcards). Еще их называют масками.

```
public interface Box<T> {
    public T get();
    public void put(T element);
}
```

```
public void unbox(Box<?> box) {
    System.out.println(box.get());
}
```

Wildcards

Подстановочные символы играют важную роль в системе типов; с их помощью можно описать тип, ограниченный некоторым семейством типов, описываемых обобщенным классом. Для обобщенного класса ArrayList, тип ArrayList<?> обозначает супертип ArrayList<T> для любого типа T.

```
public interface Box<T> {
    public T get();
    public void put(T element);
}
```

```
public class MyBox<T> implements Box<T> {
    T item;
    public void put(T element) {
        item = element;
    }
    public T get() {
        return item;
    }
}
```

```
public class Main {
    public static void unbox(Box<?> box) {
        System.out.println(box.get());
    }
    public static void main(String[] args) {
        MyBox<Integer> mb=new MyBox<>();
        mb.put( element: 12);
        unbox( box: mb);
    }
}
```

Коллекции

Коллекция – это объект-контейнер, включающий группу, как правило, однотипных объектов. Структура коллекций (collections framework) Java стандартизирует способ, с помощью которого программы хранят и обрабатывают группы объектов.

Структура коллекций:

- Интерфейсы.
- Реализации.
- Алгоритмы.

Коллекции

Преимущества использования структуры коллекций:

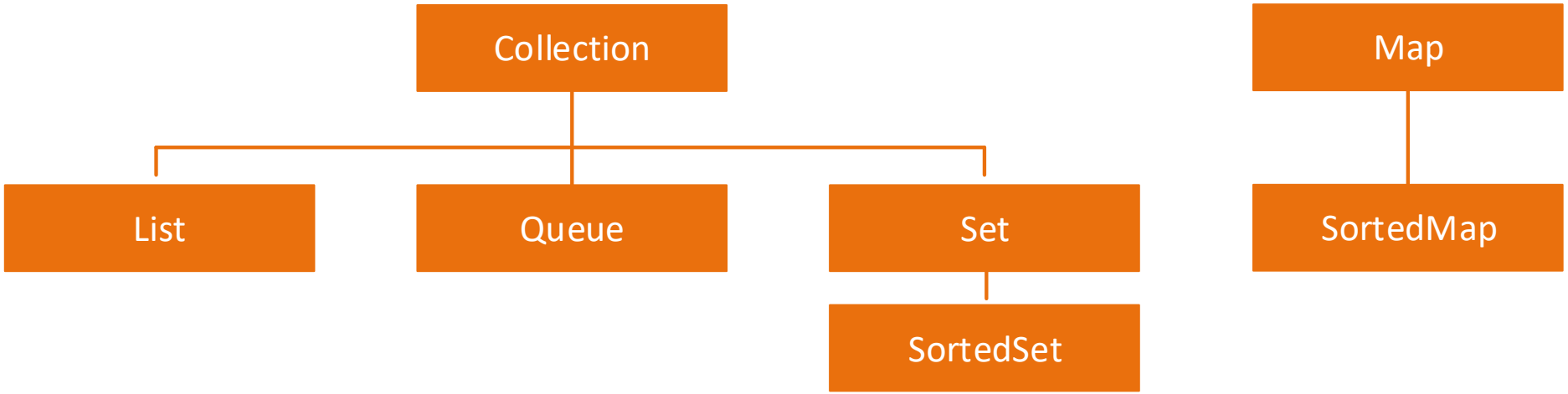
- Избавление от рутинных операций по кодированию стандартных структур данных и алгоритмов.
- Высокая эффективность реализации.
- Универсальность и простота изучения (различные типы коллекций работают похожим друг на друга образом и с высокой степенью способности к взаимодействию).
- Расширяемость.

Интерфейсы

Структура коллекций находится в пакете **java.util.***

Все коллекции в Java являются параметризованными:

```
public interface Collection<E>...
```



Интерфейсы

Корень иерархии Collection задает самые общие методы для работы с коллекциями

```
public interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);
    boolean remove(Object element);
    Iterator<E> iterator();
    ...

    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c);
    boolean removeAll(Collection<?> c);
    boolean retainAll(Collection<?> c);
    void clear();
    Object[ ] toArray();
}
```

Перемещение

Первый способ:

```
for (Object o : collection)
        System.out.println(o);
```

Второй способ:

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove();
}
```

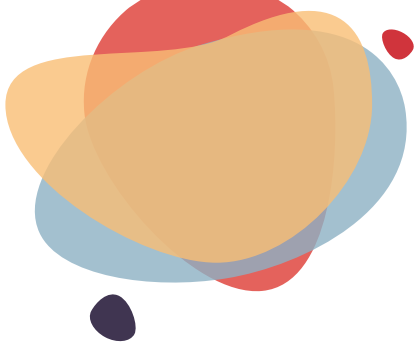
Перемещение

Первый способ:

```
Collection<String> cs = new ArrayList<String>();
cs.add("1");
cs.add("2");
cs.add("3");
for (String str : cs) {
    System.out.println(str);
}
```

Второй способ:

```
Collection<String> cs = new ArrayList<String>();
cs.add("1");
cs.add("2");
cs.add("3");
Iterator it = cs.iterator();
while (it.hasNext())
    System.out.println(it.next());
```



Перемещение

Метод remove() может быть вызван только один раз после вызова метода next(), иначе бросается исключение.

Метод remove() единственный безопасный способ модификации коллекции.

Перемещение

Метод remove() может быть вызван только один раз после вызова метода next(), иначе бросается исключение.

Метод remove() единственный безопасный способ модификации коллекции.

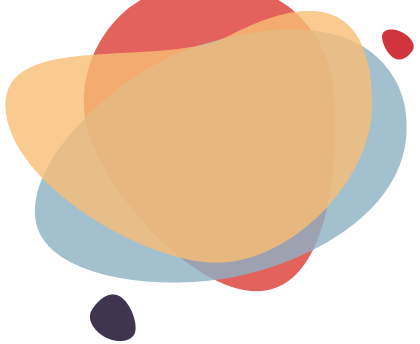
```
while(!cs.isEmpty()) {
    Iterator it = cs.iterator();
    Object o = it.next();
    System.out.println("Удаляем:" + o);
    it.remove();
}
```

Обзор коллекций

Set – коллекция без повторяющихся элементов (математическое множество). Методы совпадают с Collection но add() вернет false, если элемент уже есть в коллекции.

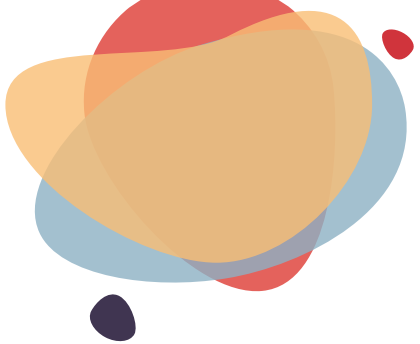
```
import java.util.*;

public class FindDups {
    public static void main(String[ ] args) {
        Set<String> s = new HashSet<String>();
        for (String a : args)
            if (!s.add(a))
                System.out.println("Duplicate detected: " + a);
        System.out.println(s.size() + " distinct words: " + s);
    }
}
```



Обзор коллекций

Интерфейс SortedSet из пакета java.util, расширяющий интерфейс Set, описывает упорядоченное множество, отсортированное по естественному порядку возрастания его элементов или по порядку, заданному реализацией интерфейса Comparator. Элементы не нумеруются, но есть понятие первого, последнего, большего и меньшего элемента.

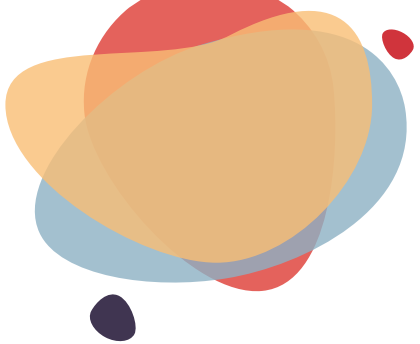


Обзор коллекций

Интерфейс List из пакета java.util, расширяющий интерфейс Collection, описывает методы работы с упорядоченными коллекциями. Иногда их называют последовательностями (sequence). Элементы такой коллекции пронумерованы, начиная от нуля, к ним можно обратиться по индексу. В отличие от коллекции Set элементы коллекции List могут повторяться.

Обзор коллекций

Интерфейс Map из пакета java.util , описывает коллекцию, состоящую из пар “ключ – значение”. У каждого ключа только одно значение, что соответствует математическому понятию однозначной функции или отображения. Такую коллекцию часто называют еще словарем (dictionary) или ассоциативным массивом (associative array).



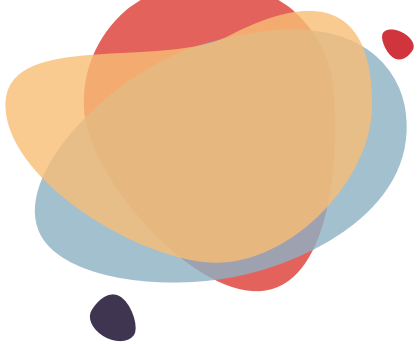
Обзор коллекций

Интерфейс SortedMap, расширяющий интерфейс Map, описывает упорядоченную по ключам коллекцию Map. Сортировка производится либо в естественном порядке возрастания ключей, либо в порядке, описываемом в интерфейсе Comparator.

Обзор коллекций

Сортировка может быть сделана только в упорядочиваемой коллекции, реализующей интерфейс List.

```
public class Sort {
    public static void main(String[ ] args) {
        List<String> list = Arrays.asList(args);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

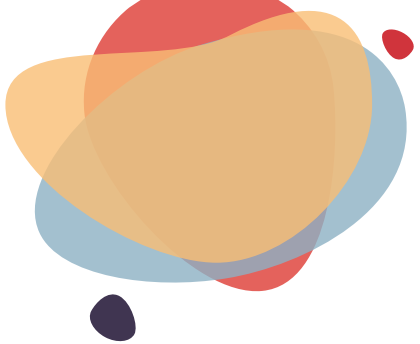


Потоки

Потоки и процессы – это абстракции управления выполнением программы.

Процесс – единица исполнения программы, обладает собственным адресным пространством, служебными структурами. Изолирован в системе от остальных процессов.

Поток – единица исполнения внутри процесса. Потоки не обладают адресным пространством, но могут исполняться параллельно друг другу.



Потоки

Каждый процесс имеет хотя бы один выполняющийся поток. Тот поток, с которого начинается выполнение программы, называется главным.

В языке Java, после создания процесса, выполнение главного потока начинается с метода main().

Затем, по мере необходимости, в заданных разработчиком местах, и при выполнении заданных им же условий, запускаются другие, побочные потоки.

Запуск потока интерфейс **Runnable**

```java
public class SomeThing implements Runnable {
    // Этот метод будет выполняться в побочном потоке
    public void run() {
        System.out.println("Привет из побочного потока!");
    }
}
```

```java
public class Main {
    static SomeThing sth;
    public static void main(String[] args)
    {
        sth = new SomeThing();
        Thread t = new Thread( target: sth);
        t.start();
        System.out.println("Главный поток завершён...");
    }
}
```

```
Главный поток завершён...
Привет из побочного потока!
```

Запуск потока наследование от Thread

```java
public class MyThread extends Thread {
    public void run() {
        System.out.println("Привет из побочного потока!");
    }
}
```

```java
public class Main {
    static MyThread t;
    public static void main(String[] args) {
        t = new MyThread(); // Создание потока
        t.start(); // Запуск потока

        System.out.println("Главный поток завершён...");
    }
}
```

Управление работой потока

Для управления потоком можно использовать методы:

- Thread.~~stop()~~ – прерывает выполнение.
- Thread.~~suspend()~~ – приостанавливает выполнение.
- Thread.~~resume()~~ – возобновляет выполнение.
- Thread.sleep() – ожидание (в мс.)
- Thread.yield() – принудительный возврат управления.
- Thread.currentThread() – возвращение родительского потока.

Управление работой потока

В настоящее время методы ~~stop()~~, ~~suspend()~~, ~~resume()~~ объявлены устаревшими и их использование не поощряется.

Вместо них рекомендуется использовать wait(), notify().

Управление выполнением

Одним из способов управлять потоками является ожидание завершения потомков со стороны родительского потока.

Если вызывать в главном потоке t.join(), то он будет ожидать завершения работы t.

```
try {
    SomeThing sth = new SomeThing();
    Thread t = new Thread( target: sth);
    t.start();
    t.join();
}
catch(InterruptedException ex) {
    System.out.println("Break!");
}
```

Синхронизация

Синхронизация используется для согласованной работы нескольких потоков. Необходимость в ней возникает, например, при попытке одновременного доступа к некоторому ресурсу. Если объявить метод как synchronized, то при передаче управления, он будет выполнен целиком. Никакой другой поток не сможет вызвать его до окончания работы.

Состояние гонок – одновременный вызов в потоках одного и того же метода для одного и того же объекта.

Существуют два основных способа синхронизации:

- Синхронизация метода.
- Синхронизация объекта.

Взаимодействие потоков

Взаимодействие потоков осуществляется с помощью методов:

- wait() – вынуждает вызывающий поток уступить монитор и перейти в состояние ожидания, пока другой поток не вызовет метод notify().
- notify() – возобновляет исполнение потока, из которого был вызван wait().
- notifyAll() – возобновляет исполнение всех потоков, из которых был вызван метод wait().

Все методы объявлены в классе Object.