Базовые и утилитные классы Java

Лекция 6

УНЦ «Инфоком»
Самара
2013

NetCracker®

План лекции

- Пакет `java.lang` и базовые классы
- Классы `Class` и `Object`
- Классы-обертки примитивных типов
- Класс `Math`
- Классы `String` и `StringBuffer`
- Пакет `java.util` и его классы
- Коллекции

Пакет java.lang

Импортируется по умолчанию (неявно)

- Класс `Class`
- Класс `Object`
- Классы-обертки примитивных типов
- Класс `Math`
- Классы `String` и `StringBuffer`
- Класс `Throwable`, базовый класс исключений
- Классы `Thread` и `ThreadGroup`
- Прочие фундаментальные типы
  - `System`, `Runtime`, `Process`, `ClassLoader`, `SecurityManager`, `Compiler`, `Cloneable`, `Comparable`

Класс Class

- Является метаклассом для всех классов
- Экземпляры содержат описания классов, загружаемых JVM
- Не имеет доступного конструктора
- Содержит методы для работы с классами и их методами
- Лежит в основе т.н. "рефлексии"

Класс Class

- При загрузке JVM файла .class создается объект класса `Class`
- При создании любого объекта создаются
  - Сам создаваемый объект
  - Объект типа `Class` описания класса объекта
  - Объекты типа `Class` описания классов-предков
  - Объект типа `Class` описания класса `Class`
- Метод `forName(String name)` получения объекта описания типа `Class` по имени класса

Класс Object

- Является суперклассом для всех классов (включая массивы)
- Переменная этого типа может ссылаться на любой объект (но не на переменную примитивного типа)
- Его методы наследуются всеми классами
- Реализует базовые операции с объектами

Методы класса Object

- Получение строкового представления объекта
  `String toString()`
- Получение ссылки на описание класса объекта
  `final Class getClass()`
- Клонирование объекта (получение копии)
  `protected Object clone()`
- Проверка равенства объектов
  `boolean equals(Object obj)`
- Получение хэш-кода объекта
  `int hashCode()`
- Метод завершения работы с объектом
  `protected void finalize()`
- Методы обслуживания блокировок в многопоточных приложениях
  `void wait(...), void notify(), void notifyAll()`

Клонирование объектов

- Считается, что результатом клонирования является копия объекта
- Массивы поддерживают операцию клонирования

```
int[] arrayCopy = (int []) array.clone();
```

- В классе `Object` метод `clone()` является защищенным
- Метод `clone()` реализуется в конкретном классе
- Никто не гарантирует того, что результатом его выполнения будет копия объекта, и даже того, что новый объект будет того же класса
- Однако существует ряд соглашений, регламентирующих реализацию метода `clone()`

Простое клонирование объектов, ряд соглашений

- Класс должен переопределять метод `clone()`
- Класс должен реализовывать интерфейс-маркер `Cloneable`
- Результат клонирования должен быть получен вызовом `super.clone()`
- Результатом работы метода `clone()` должна быть точная копия объекта

```
public Object clone() {
    Object result = null;
    try {
        result = (ThisCurrentType) super.clone();
    } catch (CloneNotSupportedException ex) { }
    return result;
}
```

Особенности клонирования

```
int[][] a = {{1, 2, 3}, {4, 5, 6}};
int[][] b = (int[][]) a.clone();
System.out.println(a[0][0]);
b[0][0] = 9;
System.out.println(a[0][0]);
```

```
1
9
```

- В результате клонирования скопировалась ссылка на объект **a**, но не объект, с которым связана ссылка
- При использовании результатов клонирования необходимо явное приведение типа
  - Начиная с Java5 для массивов можно не выполнять явное приведение типа, но только для массивов

Глубокое клонирование объектов

- Простого клонирования может быть недостаточно, если объект содержит ссылки на агрегированные объекты
- В этом случае после процедуры простого клонирования необходимо создать и их копии тоже

```
public Object clone() {
  Object result = null;
  try {
    result = (...) super.clone();
    result.a = (...) a.clone();
    ...
  } catch (CloneNotSupportedException ex) { }
  return result;
}
```

Равенство объектов

- Простого сравнения ссылок недостаточно для сравнения содержимого объектов
- Для сравнения объектов по их содержимому применяется метод `equals(Object obj)`
- В классе `Object` метод реализован таким образом, что возвращает `true` только при сравнении с самим объектом
- Конкретный класс может переопределять метод `equals(Object obj)`

Равенство объектов

- Метод `equals(...)` должен проверять эквивалентность объектов с точки зрения бизнес-логики
- Отношение, задаваемое на множестве объектов этим методом, должно обладать следующими свойствами:
  - рефлективность (`x.equals(x)`)
  - симметричность (`x.equals(y)` и `y.equals(x)`)
  - транзитивность (`x, y, z`)
  - непротиворечивость (многократное `x.equals(y)`)
  - сравнение с `null` (`false`)

Равенство объектов

```
public class Gadget implements Item {
    private int p1;
    private double p2;
    ...
    public boolean equals(Object object) {
        if (object == this) {
            return true;
        }
        if (!(object instanceof Gadget)) {
            return false;
        }
        Gadget gdgt = (Gadget) object;
        return (this.p1 == gdgt.p1) && (this.p2 == gdgt.p2);
    }
}
```

Хэш-код объекта

- Метод `int hashCode()` предназначен для получения хэш-кода – числа, используемого для быстрого сравнения объектов
- Если объект не изменял свое состояние, то значение хэш-кода не должно изменяться
- Если два объекта эквивалентны (с точки зрения метода `equals()`), то хэш-коды объектов должны быть одинаковыми
- Если хэш-коды объектов одинаковы, то это еще не значит, что объекты эквивалентны
- Изменение реализации в классе метода `equals()` влечет за собой изменение реализации метода `hashCode()`

Хэш-код объекта

```
public class Employee {
  int         employeeId;
  String      name;
  Department dept;

  //some other methods

  public int hashCode() {
    int hash = 42;
    hash = hash * 17 + employeeId;
    hash = hash * 31 + name.hashCode();
    hash = hash * 13 + (dept == null ? 0 : dept.hashCode());
    return hash;
  }
}
```

Классы-обертки примитивных типов

- Значения примитивных типов не могут быть непосредственно использованы в контексте, где требуется ссылка
- Ссылочное представление значений примитивных типов является основной задачей т.н. классов-оберток
- Экземпляр такого класса хранит внутри значение примитивного типа и предоставляет доступ к этому значению

Классы-обертки примитивных типов

- Boolean
- Byte
- Character
- Double
- Float
- Integer
- Long
- Number
- Short
- Void

Задачи классов-оберток примитивных типов

- Ссылочное представление значений примитивных типов
- Хранение вспомогательных функций для работы со значениями примитивных типов
- Представление примитивных типов и их значений в механизмах рефлексии

Наполнение классов-оберток

- Константы типов
  `Integer.MAX_VALUE`, `Double.NaN`
- Конструкторы: по значению и строке
  `Float(float value)`, `Float(String s)`
- Методы получения значения
  `aBoolean.booleanValue()`, `aFloat.floatValue()`
- Методы преобразования в значение примитивного типа
  `Integer.parseInt(String s)`, `aFloat.byteValue()`
- Методы преобразования в объект класса-обертки
  `Integer.valueOf(String s, int radix)`
- Методы проверки состояния и вида значения
  `aWrapedValue.compareTo(...)`, `aDouble.isInfinite()`
- Специальные методы, обусловленные спецификой типа
  `Double.longBitsToDouble(...)`, `Integer.toHexString()`

Классы-обертки примитивных типов

- Каждому примитивному типу сопоставлен соответствующий класс-обертка
- Все классы-обертки имеют публичный конструктор (кроме класса `Void`)
- Объекты классов-оберток могут сравниваться между собой методом `equals()`
- Значения примитивных типов можно получить из объектов вызовом методов `<type>Value()`
- Классы-обертки предоставляют статические методы работы с примитивными типами

Класс Math

- Предназначен для выполнения простых математических операций
- Не имеет явного конструктора
- Является final-классом
- Все методы являются статическими
- Не гарантирует повторяемости результатов на различных платформах (в отличие от класса `StrictMath`)

Наполнение класса Math

- Константы `E` и `PI`
- Функции взятия модуля `abs()`
- Функции максимума и минимума `max()`, `min()`
- Функции округления `round()`, `rint()`
- Функции ближайшего целого `ceil()`, `floor()`
- Тригонометрические функции `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()`
- Функции перевода `toDegrees()`, `toRadians()`, `atan2()`
- Функции степени `pow()`, `exp()`, `log()`, `sqrt()`
- Случайное значение `random()` (см. класс `java.util.Random`)

Хранение строк

- `byte []`
  Массив байт кодов
- `char []`
  Массив Unicode-символов
- `String`
  Неизменяемая строка
- `StringBuffer`
  Изменяемая строка

Работа со строками. Класс String

- Значение объекта класса `String` не может быть изменено без порождения нового объекта
- Реализует операции для строки в целом
- Экземпляры этого класса можно создавать без ключевого слова `new`
- Каждый строковый литерал порождает экземпляр `String`
- Значение любого типа может быть приведено к строке

Наполнение класса String

- Строковое представление
  `valueOf()`, `copyValueOf()`
- Преобразование типов
  `getBytes()`, `getChars(…)`, `toCharArray()`, `toString()`
- Сравнение
  `compareTo(…)`, `compareToIgnoreCase(…)`, `contentEquals(…)`, `equals(…)`, `equalsIgnoreCase(…)`, `intern()`
- Выделение элементов
  `charAt(…)`, `substring(…)`, `split(…)`
- Операции над всей строкой
  `concat(…)`, `replace(…)`, `replaceAll(…)`, `replaceFirst(…)`, `toLowerCase()`, `toUpperCase()`, `trim()`
- Проверка содержимого строки
  `endsWith(…)`, `indexOf(…)`, `lastIndexOf(…)`, `length()`, `matches(…)`, `regionMatches(…)`, `startsWith(…)`

Работа со строками. Класс StringBuffer

- Реализует методы модификации строки без порождения нового объекта
- Реализует операции с элементами строки по отдельности
- Используется по умолчанию при конкатенации строк
- Для хранения строк использует буфер переменного объема

Наполнение класса StringBuffer

- Добавление фрагментов
  `append(…)`, `insert(…)`
- Поиск вхождений
  `indexOf(…)`, `lastIndexOf(…)`
- Извлечение фрагментов
  `charAt(…)`, `getChars(…)`, `reverse()`, `substring(…)`
- Модификация строки
  `delete(…)`, `deleteCharAt(…)`, `replace(…)`, `setCharAt(…)`, `setLength(…)`
- Состояние буфера
  `length()`, `capacity()`, `ensureCapacity(…)`, `trimToSize()`

Конкатенация строк

```
System.out.println("a = " + a + ";");
```

```
System.out.println(
  (new StringBuffer("a = "))
  .append(a)
  .append(";")
  .toString()
);
```

- Не стоит злоупотреблять автоматической конкатенацией
- Особенно если для вас критична память и скорость выполнения программы

Системные классы

- `ClassLoader` – абстрактный класс, необходим для загрузки описания типов (объектов `Class`) в память JVM
- `SecurityManager` – реализует методы проверки допустимости запрашиваемой операции
- `System` – содержит набор полезных статических полей и методов
- `Runtime` – позволяет приложению взаимодействовать со средой исполнения
- `Process` – представляет интерфейс взаимодействия с внешней программой, запущенной через `Runtime`

Пакет java.util

- Классы для работы со временем
- Классы для работы с локализацией
- Классы для работы с массивами
- Классы и интерфейсы коллекций
- Прочие вспомогательные классы и интерфейсы

Классы работы со временем

- `Date`
  Отражает дату и время с точностью до миллисекунд. Не рекомендуется к использованию
- `Calendar` и сопутствующие
  Содержит константы и методы для работы с датой и временем с учетом особенностей локализации
- `Timer`
  Позволяет создавать задания для более позднего запуска (с использованием потоков инструкций)

Методы класса Calendar

- Установка значения поля календаря

```
public void set(int field, int value)
```

- Добавляет смещение к текущей величине поля

```
public abstract void add(int field, int
amount)
```

- Добавляет смещение к величине поля, причем не производит изменения старших полей

```
public abstract void roll(int field, boolean
up)
```

Методы класса Calendar

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMMM dd
HH:mm:ss");
Calendar cal = Calendar.getInstance();
cal.set(Calendar.YEAR, 2012);
cal.set(Calendar.MONTH, Calendar.AUGUST);
cal.set(Calendar.DAY_OF_MONTH, 31);
System.out.println("Initialy set date :" +
sdf.format(cal.getTime()));
cal.add(Calendar.MONTH, 1);
System.out.println("Month changed by add():" +
sdf.format(cal.getTime()));
cal.roll(Calendar.DATE, 45);
System.out.println("Date changed by roll():" +
sdf.format(cal.getTime()));
```

```
Initialy set date :2012 Август 31 15:05:09
Month changed by add() :2012 Сентябрь 30 15:05:09
Date changed by roll():2012 Сентябрь 15 15:05:09
```

Классы для работы с локализацией

- **`Locale`**
  Содержит константы и методы для работы с языками и особенностями регионов
- **`TimeZone`**
  Содержит методы для работы с часовыми поясами
- **`SimpleTimeZone`**
  Реализует **`TimeZone`** для Григорианского календаря

java.util.Random

- Экземпляр класса является отдельным генератором псевдослучайных чисел (ГПСЧ)
- Различные ГПСЧ позволяют формировать некоррелированные последовательности
- «Основание» имеет размерность 48bit
- Методы получения ПСЧ: `nextBoolean()`, `nextByte()`, `nextDouble()`, `nextFloat()`, `nextInt()`, `nextLong()`, `nextGaussian()`
- Метод настройки `setSeed(long seed)`

Регулярные выражения

- Позволяют сопоставлять текст с шаблоном, выполнять замену текста
- Операции осуществляются с помощью универсальных символов, которые специальным образом интерпретируются
- Используются в большом количестве языков программирования

Пакет java.util.regex

- Класс `Pattern`
  - Реализует шаблоны регулярных выражений. Позволяет составлять сложные шаблоны и разделять строки на элементы
- Класс `Matcher`
  - Реализует поиск элементов, соответствующих шаблону, в строках и проверку строк на соответствие шаблону
- `PatternSyntaxException`
  - указывает на синтаксическую ошибку в выражении

Коллекции

- Коллекции (контейнеры) – хранилища, поддерживающие разнообразные способы накопления и упорядочивания объектов с целью обеспечения возможностей эффективного доступа к ним
- В Java коллекции разделены на интерфейсы, абстрагирующие общие принципы работы с коллекциями, и классы, реализующие конкретную функциональность
- Не все методы, заявленные в интерфейсах, должны в действительности реализовываться классами. Часть методов может просто выбрасывать исключение `UnsupportedOperationException`

Интерфейс Collection

- Образующий в иерархии типов коллекций
- Определяет базовую функциональность любой коллекции
- Подразумевает добавление, удаление, выбор элементов в коллекции
- Допускает дубликаты и пустые элементы
- Абстрактный класс `AbstractCollection` определяет реализацию ряда методов

Методы интерфейса Collection

- Добавление элементов
  `boolean add(Object o),`
  `boolean addAll(Collection c)`
- Исключение элементов
  `boolean remove(Object o),`
  `boolean removeAll(Collection c),`
  `boolean retainAll(Collection c),`
  `void clear()`
- Состояние коллекции
  `boolean contains(Object o),`
  `boolean containsAll(Collection c),`
  `boolean isEmpty(),`
  `int size()`
- Вспомогательные методы
  `Object[] toArray(),`
  `Iterator iterator()`

Интерфейс Set

- Расширяет интерфейс `Collection`
- Не допускает наличие дубликатов
- Разрешает только одну ссылки `null`
- Объекты коллекции должны корректно реализовывать метод `equals()`
- Расширение `SortedSet` требует упорядоченности по значениям набора
- Абстрактный класс `AbstractSet` определяет реализацию ряда методов

Интерфейс List

- Расширяет интерфейс `Collection`
- Подразумевает хранение упорядоченной последовательности объектов
- Порядок хранения определяется порядком добавления элементов
- Позволяет обращаться к элементам по их номеру в коллекции
- Абстрактный класс `AbstractList` определяет реализацию ряда методов

Специальные методы интерфейса List

- Адресное добавление
  `void add(int index, Object o)`,
  `boolean addAll(int index, Collection c)`
- Адресные операции с элементами
  `Object get(int index)`,
  `Object set(int index, Object o)`,
  `Object remove(int index)`
- Операции поиска
  `int indexOf(Object o)`,
  `int lastIndexOf(Object o)`
- Специальные операции
  `List subList(int from, int to)`,
  `ListIterator listIterator()`

Интерфейс Iterator

- Позволяет работать с любой коллекцией как с перебираемым набором элементов
- Метод `Iteraror iterator()` вызывается для объекта коллекции
- Проверка наличия следующего объекта коллекции `boolean hasNext()`
- Следующий объект `Object next()`
- Исключение объекта `void remove()`

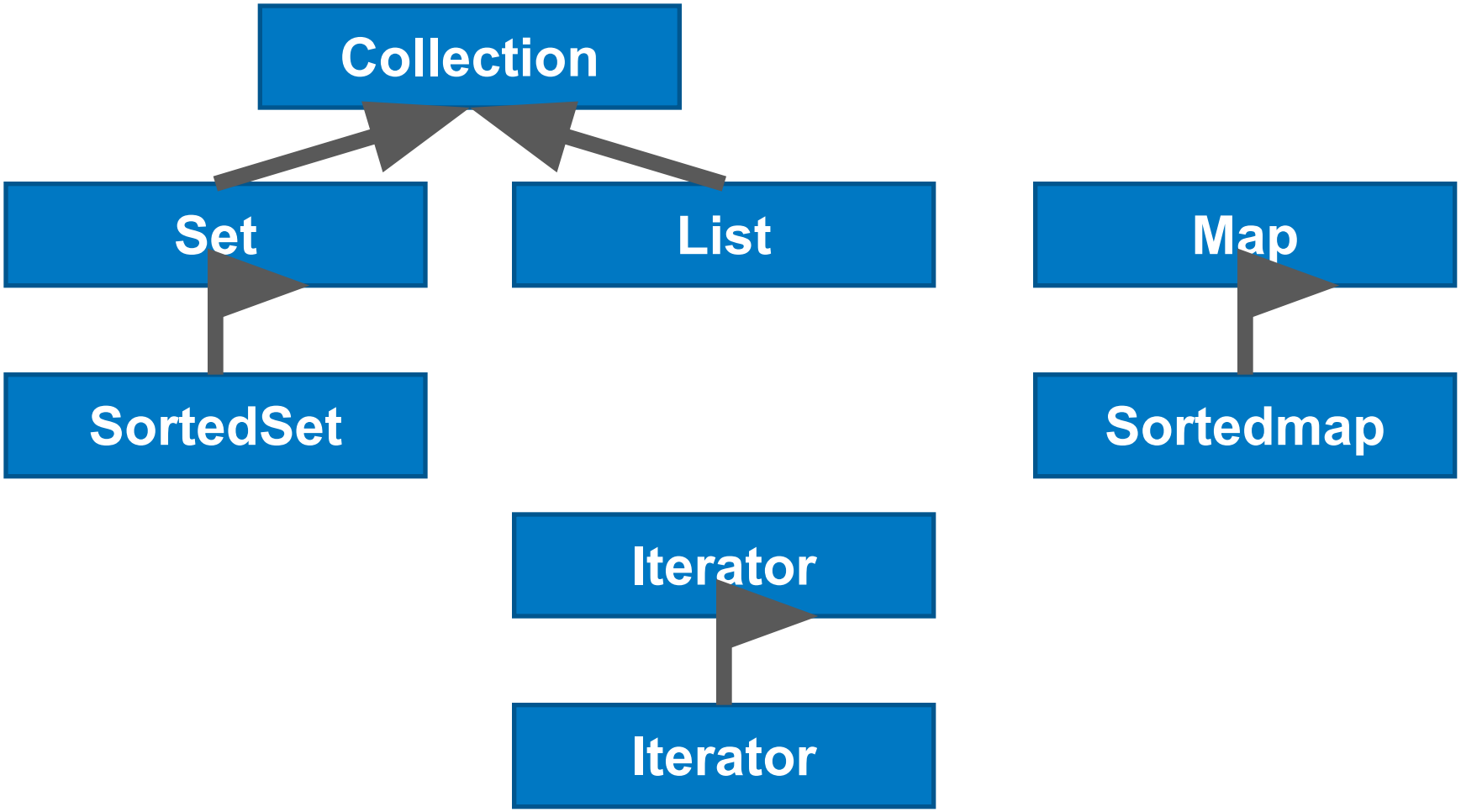
Интерфейс Map

- Не расширяет интерфейс `Collection`
- Подразумевает хранение набора объектов парами ключ/значение
- Ключи должны быть уникальными
- Порядок следования пар не определен
- Расширение `SortedMap` требует упорядоченности по значениям ключей
- Абстрактный класс `AbstractMap` определяет реализацию ряда методов

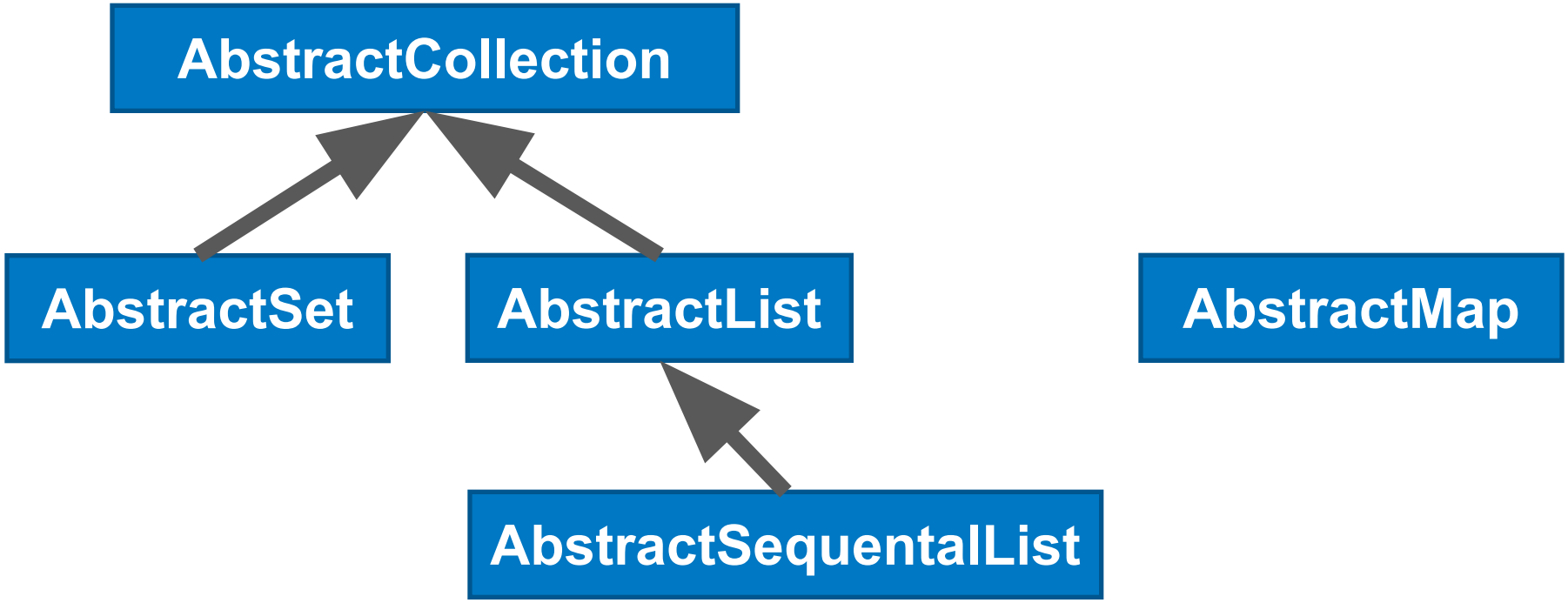
Методы интерфейса Map

- Добавление объектов
  `Object put(Object key, Object value)`,
  `void putAll(Map t)`
- Исключение объектов
  `Object remove(Object key)`,
  `void clear()`
- Доступ к объекту по ключу
  `Object get(Object key)`
- Состояние
  `boolean containsValue(Object value)`,
  `boolean containsKey(Object key)`,
  `int size()`,
  `boolean isEmpty()`
- Преобразование типа
  `Set entrySet()`,
  `Set keySet()`,
  `Collection values()`

Иерархия интерфейсов коллекций



Иерархия абстрактных классов коллекций



Классы коллекций

- Динамические массивы
  `ArrayList` (`List`), `Vector` (`List`)
- Двухсвязный список
  `LinkedList` (`List`)
- Упорядоченные множество и карта
  `TreeSet` (`Set`), `TreeMap` (`SortedMap`)
- Ряд других классов
  `HashMap` (`Map`), `HashSet` (`Set`), …

Класс java.util.ArrayList

- Расширяет класс `AbstractList`
- Динамически расширяется при добавлении новых элементов в коллекцию
- Методы доступа к элементам не синхронизированы
- Рекомендуется для использования при работе с коллекцией из одной нити

Класс java.util.ArrayList

```
ArrayList arrayList = new ArrayList();
for (int i = 0; i < 5; i++) {
  arrayList.add("Item " + i);
  System.out.println(arrayList.get(i));
}

System.out.println("---");

arrayList.add(2, "Item to insert");
Iterator it = arrayList.iterator();
while (it.hasNext()) {
  System.out.println((String) it.next());
}
```

```
Item 0
Item 1
Item 2
Item 3
Item 4
---
Item 0
Item 1
Item 2
Item to insert
Item 3
Item 4
```

Класс java.util.LinkedList

- Реализует интерфейс `List`
- Является реализацией двусвязного списка
- Добавлены дополнительные методы доступа, добавления и удаления элементов в начало и конец списка
- Удобен для организации стека

Класс java.util.LinkedList

```
LinkedList linkedList = new LinkedList();
linkedList.add("Item 1");
linkedList.add("Item 2");
linkedList.addFirst("Item 3");
linkedList.addLast("Item 4");
printCurrentCollection(linkedList);

linkedList.remove(1);
printCurrentCollection(linkedList);

linkedList.remove(linkedList.getLast());
printCurrentCollection(linkedList);
```

```
Item 3
Item 1
Item 2
Item 4
---
Item 3
Item 2
Item 4
---
Item 3
Item 2
```

Класс java.util.Hashtable

- Реализует интерфейс `Map`
  - и расширяет устаревший класс `Dictionary`
- Хранит объекты в виде пар ключ/значение
- Не позволяет `null` в ключах и значениях
- Использует алгоритм хэширования для увеличения скорости доступа к данным
- Число `key.hashCode() % array.length` используется для определения индекса элемента
- Синхронизированный доступ

Класс java.util.HashMap

- Расширяет класс `AbstractMap`
- Похож на класс `Hashtable`
- Хранит объекты в виде пар ключ/значение
- Для одного ключа и элементов допускаются значения типа `null`
- Порядок хранения элементов не совпадает с порядком их добавления и может меняться
- Обеспечивает постоянное время доступа для операций `get()` и `put()`

Класс java.util.HashMap

```
HashMap hashmap = new HashMap();
for (int i = 0; i < 10; i++) {
  hashmap.put("Key" + i, new Integer(i));
}
System.out.println("Key1: " +
    (Integer) hashmap.get("Key1"));
System.out.println("---");

Map.Entry m = null;
Iterator iterator =
hashmap.entrySet().iterator();
while (iterator.hasNext()) {
  m = (Map.Entry) iterator.next();
  System.out.println(m.getKey() + ": " +
       (Integer) m.getValue());
}
```

```
Key1: 1
---
Key0: 0
Key2: 2
Key1: 1
Key4: 4
Key3: 3
Key6: 6
Key5: 5
Key8: 8
Key7: 7
Key9: 9
```

Класс java.util.TreeMap

- реализует **SortedMap**
- расширяет **AbstractMap**
- содержит ключи в порядке возрастания
- запрещено применение **null** для ключей
- при использовании дубликатов ключей ссылка на предыдущий объект теряется

```
TreeMap tm = new TreeMap();
tm.put("key", "String1");
tm.put("key", "String2"); // изменение ссылки по ключу key
```

Класс Collections

- Утилитный класс
- Содержит ряд статических методов прикладного назначения, позволяющих оперировать объектами коллекций
- Группы методов:
  - Создание и поддержка оберток коллекций
  - Прочие прикладные методы

Обертки коллекций

- Синхронизированные
  - Обеспечивают механизмы синхронизации доступа для многопоточных приложений
  - `List synchronizedList(List l)`, …
- Неизменяемые
  - Запрещают использование методов модификации значений
  - `Map unmodifiableMap(Map m)`, …

Прикладные методы

- Методы поиска минимума и максимума
  `min()`, `max()`
- Работа со списками
  `reverse()`, `shuffle()`, `fill()`, `copy()`, `nCopies()`
- Сортировка списков
  `sort()`
- Поиск элементов в списке
  `binarySearch()`
- Прочие прикладные методы

java.util.Arrays

Содержит статические методы для работы с массивами

- Представление массива списком
  `List asList(Object[] a)`
- Поиск элемента в массиве
  `int binarySearch(…[] a, … key)`
- Сравнение массивов по элементам
  `boolean equals(…[] a1, …[] a2)`
- Заполнение массива элементами
  `fill(…[] a, int from, int to, … val)`
- Сортировка массива
  `sort(…[] a, int from, int to)`

Спасибо за внимание!

NetCracker®

Дополнительные источники

- Арнолд, К. Язык программирования Java [Текст] / Кен Арнолд, Джеймс Гослинг, Дэвид Холмс. – М. : Издательский дом «Вильямс», 2001. – 624 с.
- Вязовик, Н.А. Программирование на Java. Курс лекций [Текст] / Н.А. Вязовик. – М. : Интернет-университет информационных технологий, 2003. – 592 с.
- Хорстманн, К. Java 2. Библиотека профессионала. Том 1. Основы [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010 г. – 816 с.
- Хорстманн, К. Java 2. Библиотека профессионала. Том 2. Тонкости программирования [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010 г. – 992 с.
- Эккель, Б. Философия Java [Текст] / Брюс Эккель. – СПб. : Питер, 2011. – 640 с.
- JavaSE at a Glance [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/overview/index.html, дата доступа: 21.10.2011.
- JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/documentation/api-jsp-136079.html, дата доступа: 21.10.2011.