Теория и технология программирования

Программирование на языке Java

Лекция 4. Проектирование классов в Java

Глухих Михаил Игоревич, к.т.н., доц.
mailto: glukhikh@mail.ru

Статически вложенные классы

- В некоторых случаях в проекте появляется некоторый класс А, использование которого жестко привязано к классу Б (при этом класс А играет вспомогательную роль)
- Примеры
  - Линейный список и Узел
  - Таблица и Строка таблицы

Статически вложенные классы

- Для подобных случаев в Java предусмотрены классы, определения которых вкладываются в другие классы (nested classes), например

```
public final class Integer {
    // ...
    // Статически вложенный класс
    private static class IntegerCache {
        // ...
    }
}
```

Спецификаторы вложенных классов

- public – вложенным классом можно пользоваться во всей программе (его полное имя выглядит так: java.lang.Integer.IntegerCache)
- private – вложенным классом можно пользоваться только во внешнем классе
- static – статически вложенный класс; объект вложенного класса не имеет неявной информации об объекте внешнего класса (**если** такая информация не требуется, класс **всегда** следует вкладывать статически)

Определение класса кэш целых. Статические инициализаторы

```
private static class IntegerCache {
    static final int low = -128;
    static final int high = 127;
    static final Integer cache[];
    // Статический инициализатор
    static {
        cache = new Integer[(high - low) + 1];
        int j = low;
        for (int k = 0; k < cache.length; k++)
            cache[k] = new Integer(j++);
    }
    private IntegerCache() {}
}
```

Статические инициализаторы

- Выполняются в начале программы, поочередно для всех классов
- Порядок выполнения определяется автоматически

Использование класса кэш целых. Статические методы-создатели

```
public final class Integer {
   private static class IntegerCache {
      // ...
   }
   // Статический метод-создатель
   // (static factory method)
   public static Integer valueOf(int i) {
      if (i >= IntegerCache.low &&
          i <= IntegerCache.high)
         return IntegerCache.cache[i – IntegerCache.low];
      else
         return new Integer(i);
   }
}
```

Статические методы-создатели и конструкторы

- Использование конструктора

```
Integer i = new Integer(20);
```

- Использование статического метода-создателя

```
Integer i = Integer.valueOf(20);
```

Статические методы-создатели и конструкторы

- Преимущества статических методов-создателей
  - можно выбрать подходящее имя
  - не требуется создавать новый объект каждый раз
  - при желании могут вернуть объект подкласса

Внутренние и локальные классы

- Внутренние (inner) классы – нестатически вложенные классы
  - объект внутреннего класса несет в себе неявную ссылку на объект внешнего класса (за счет этого можно пользоваться данными и методами внешнего класса)
- Локальные (local) классы – определяются внутри одного из методов внешнего класса
- Примеры будут рассмотрены позже

Исключения

- Объект "исключение" содержит информацию о произошедшей ошибке
- При возникновении исключения мы поднимаемся наверх по стеку вызовов функций, пока не найдем блок обработки исключения
- Исключения используются в тех случаях, когда функция не может адекватно обработать возникшую ситуацию (см. IntegerInterval.conj)

Работа с исключениями в Java

- Конструкция формирования

```
throw new IllegalArgumentException(
   "Аргумент неправильный!");
```

- Блок обработки исключения

```
try {
} catch (IllegalArgumentException ex) {
} catch (UnsupportedOperationException ex) {
} catch (RuntimeException ex) {
} catch (Exception ex) {
} finally {
}
```

- Признак возможности возникновения исключения

```
public void f() throws SomeException { ... }
```

Совместная обработка исключений в JDK7

```
try {
} catch (IllegalArgumentException|
         UnsupportedOperationException ex) {
} finally {
}
```

Блок finally

- Всегда выполняется после окончания блока try/catch:
  - в случае успешного завершения блока try
  - в случае успешной обработки одного из исключений в блоке catch
  - в случае возникновения необрабатываемого исключения
- Используется обычно для выполнения определенных завершающих действий (например, закрытия файла)

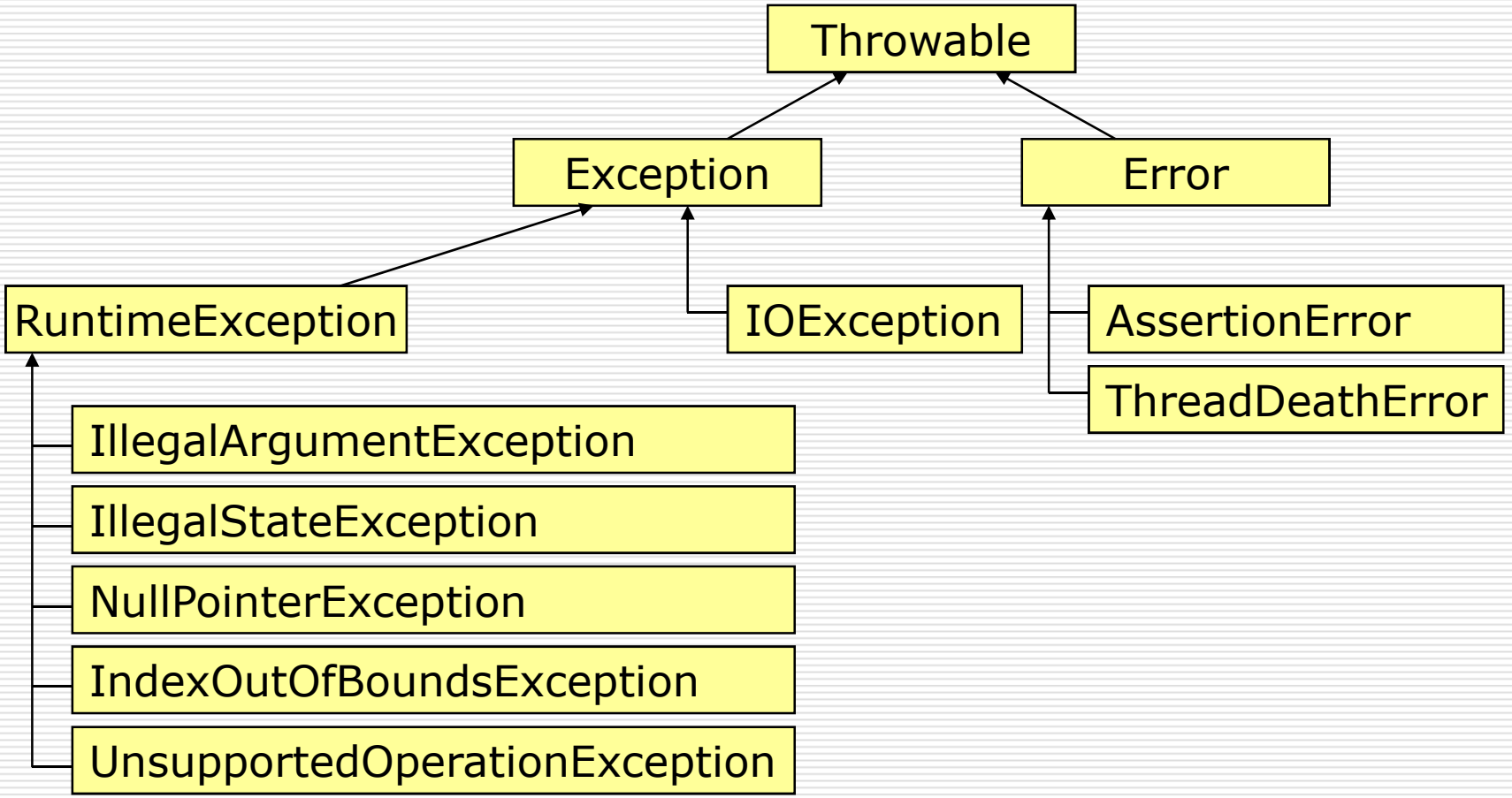
Закрытие ресурсов в JDK 6

```
static String readFirstLineFromFile(
    String path) throws IOException {
    BufferedReader br = new BufferedReader(
        new FileReader(path));
    try {
        return br.readLine();
    } finally {
        if (br != null) br.close();
    }
}
```

Закрытие ресурсов в JDK 7

```
static String readFirstLineFromFile(
    String path) throws IOException {
    try (BufferedReader br =
            new BufferedReader(
                new FileReader(path))) {
        return br.readLine();
    }
}
```

Иерархия исключений в Java



Иерархия исключений в Java

- Throwable – любое исключение
  - Error – исключения, которые обычно не обрабатывают
    - AssertionError – не сработала программная проверка
  - Exception – исключения, которые могут обрабатываться
    - RuntimeException – неконтролируемые исключения (можно не указывать throws)

Основные методы исключений

- Конструкторы

```
new Exception("Описание")
new Exception(throwable)
new Exception("Описание", throwable)
```

- assert условие : "Описание" – программная проверка
- getMessage() – получить сообщение
- printStackTrace() – распечатать стек вызовов

Перечисления в Java

- Предназначены для реализации типов с ограниченным количество значений
- Пример:
  тип "планета солнечной системы"
  - Меркурий, Венера, Земля, ...

Простое определение перечисления

```
public enum Planet {
    // Элементы перечисления
    MERCURY,
    VENUS,
    EARTH,
    MARS,
    JUPITER,
    SATURN,
    URANUS,
    NEPTUN;
}
```

Использование перечисления

```
Planet p1 = Planet.EARTH;
// или
switch (p1) {
    case MERCURY:
        // ...
        break;
    case MARS:
        // ...
        break;
    default:
        break;
}
```

Общие методы перечислений

- На самом деле перечисления – это классы, наследующие класс Enum

```
// Перебрать все планеты
for (Planet p: Planet.values()) {
    // Вывести числовой код
    System.out.println(p.ordinal());
}
// Определить планету по строке-названию
Planet p = Planet.valueOf("EARTH");
```

Дополнительные методы перечислений

```
public enum Planet {
    MERCURY(3.302e+23, 2.439e+06),
    // ...
    NEPTUNE(1.024e+26, 2.477e+07);
    private final double mass;
    private final double radius;
    private final double gravity;
    private static final double G = 6.67300e-11;
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
        this.gravity = G * mass / (radius * radius);
    }
}
```

Что такое перечисление и его элементы в Java?

- Перечисление – это тоже класс, наследник класса Enum
- Элемент перечисления – это статическое поле соответствующего класса, тип поля совпадает с типом перечисления

Перечисления в C/C++ и Java

- C/C++
  - перечисления хранятся как целые числа
- Java
  - перечисления хранятся как ссылки на объекты класса (и поэтому могут иметь методы, поля и так далее)

Понятие интерфейса

- Интерфейс определяет **действия**, которые должен уметь выполнять класс, **реализующий** данный интерфейс
- В интерфейсе перечисляются методы, которые должны быть реализованы соответствующим классом (как правило, они открытые)
- Определения методов в интерфейсе не указываются, после заголовка ставится точка с запятой
- Также могут быть статические константы public static final
- В интерфейсе не бывает конструкторов
- Интерфейс напоминает **чисто абстрактный класс**

Пример простого интерфейса

- Интерфейс **сравниваемый** – данный объект можно сравнить с другим объектом определенного типа

```
public interface Comparable<T> {
    /**
     * @param o the object to be compared
     * @return a negative integer, zero, or
     * a positive integer as this object is
     * less than, equal to, or greater than
     * the specified object
     */
    public int compareTo(T o);
}
```

Контракты методов

- Помимо того, что в интерфейсе перечисляются требуемые методы (формальная сторона)
- В нем также описываются, как эти методы должны работать (compareTo должна возвращать нуль при this = o, положительное число при this > o, отрицательное число при this < o, должны соблюдаться ряд свойств)
- Подобное описание называется контрактом метода (general contract)

Что такое <T> и просто T?

- При реализации интерфейса вместо T следует подставить тот тип, с которым мы умеем сравниваться
- Технология напоминает **шаблоны (templates)** в языке C++
- В Java, интерфейсы и классы с возможностью подобной настройки называются **generic**
- Тип T – **всегда** ссылочный

Как реализовать интерфейс в классе?

- В определении класса следует указать implements Comparable<T>, при этом вместо T подставляется требуемый тип
- В классе должна быть **реализована** функция compareTo
- Класс может реализовывать любое количество интерфейсов (например, Comparable, Cloneable)

Пример

```
public class Rational implements Comparable<Rational>
    {
     private final int num;
     private final int denom;
     // ...
     @Override
     public int compareTo(Rational o) {
         Rational res = this.sub(o);
         if (res.num > 0)
             return 1;
         else if (res.num == 0)
             return 0;
         else
             return -1;
     }
}
```

Зачем реализовывать интерфейс?

- Реализация интерфейса позволяет использовать код такого типа

```
Rational r;
// Любой объект, который реализует
// интерфейс Comparable<Rational>
Comparable<Rational> s;
// инициализация …
if (s.compareTo(r) > 0)  { … }
Object obj;
// инициализация …
if (obj instanceof Comparable) { … }
```

Функция сортировки на основе Comparable

```
public class Arrays {
   public void sort(Object[] a) {
      // ...
      // Встречаются конструкции следующего вида
      if (((Comparable)a[i-1]).compareTo(a[i])<=0) {
         // Что-нибудь сделать
      }
   }
}
// Что будет, если объекты массива – не Comparable?
```

Интерфейс Cloneable (JavaDoc)

- public interface **Cloneable**
- A class implements the Cloneable interface to indicate to the Object.clone() method that it is legal for that method to make a field-for-field copy of instances of that class.
- Invoking Object's clone method on an instance that does not implement the Cloneable interface results in the exception CloneNotSupportedException being thrown.
- By convention, classes that implement this interface should override Object.clone (which is protected) with a public method. See Object.clone() for details on overriding this method.

Интерфейс Cloneable

- Не включает в себя ни одной функции (так называемый marker interface)
- Object.clone() проверяет, реализует ли данный объект Cloneable (если нет, бросает исключение)
- Плюс имеется соглашение (неконтролируемое!) о том, что классы, реализующие Cloneable, переопределяют clone() как открытый метод

Итоги

- Классы, поля, методы, права доступа
- Вложенные классы
- Исключения
- Перечисления
- Интерфейсы