Объекты, классы и пакеты в Java

Алексей Владыкин

22 сентября 2014


"Object-oriented programming
is an exceptionally bad idea
which could only have
originated in California."
-- Edsger Dijkstra

Определение ООП

> Объект — это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области
>
> *Гради Буч*

- Объектно-ориентированное программирование — парадигма программирования, в которой программа строится из взаимодействующих объектов
- Ср.: процедурное, функциональное, логическое программирование

Свойства объекта

- Объект является экземпляром класса
- Объект имеет внутреннее состояние
- Объект может принимать сообщения
  (в большинстве языков сообщение = вызов метода)
- Объект — это «умные данные»

Возможности ООП

- Инкапсуляция
  Сокрытие деталей реализации за внешним интерфейсом
- Наследование
  Создание производных классов, наследующих свойства базового
- Полиморфизм
  Разная обработка сообщений в разных классах

ООП в Java

- Инкапсуляция, наследование и полиморфизм поддерживаются на уровне языка
- В Java все является объектом, кроме примитивных типов
- Исполняемый код может находиться только в классе
- Стандартная библиотека предоставляет огромное количество классов, и можно создавать свои

Зачем нужны пакеты

- Задание пространства имен,
  предотвращение коллизий имен классов
- Логическая группировка связанных классов
- Инкапсуляция

Как работают пакеты

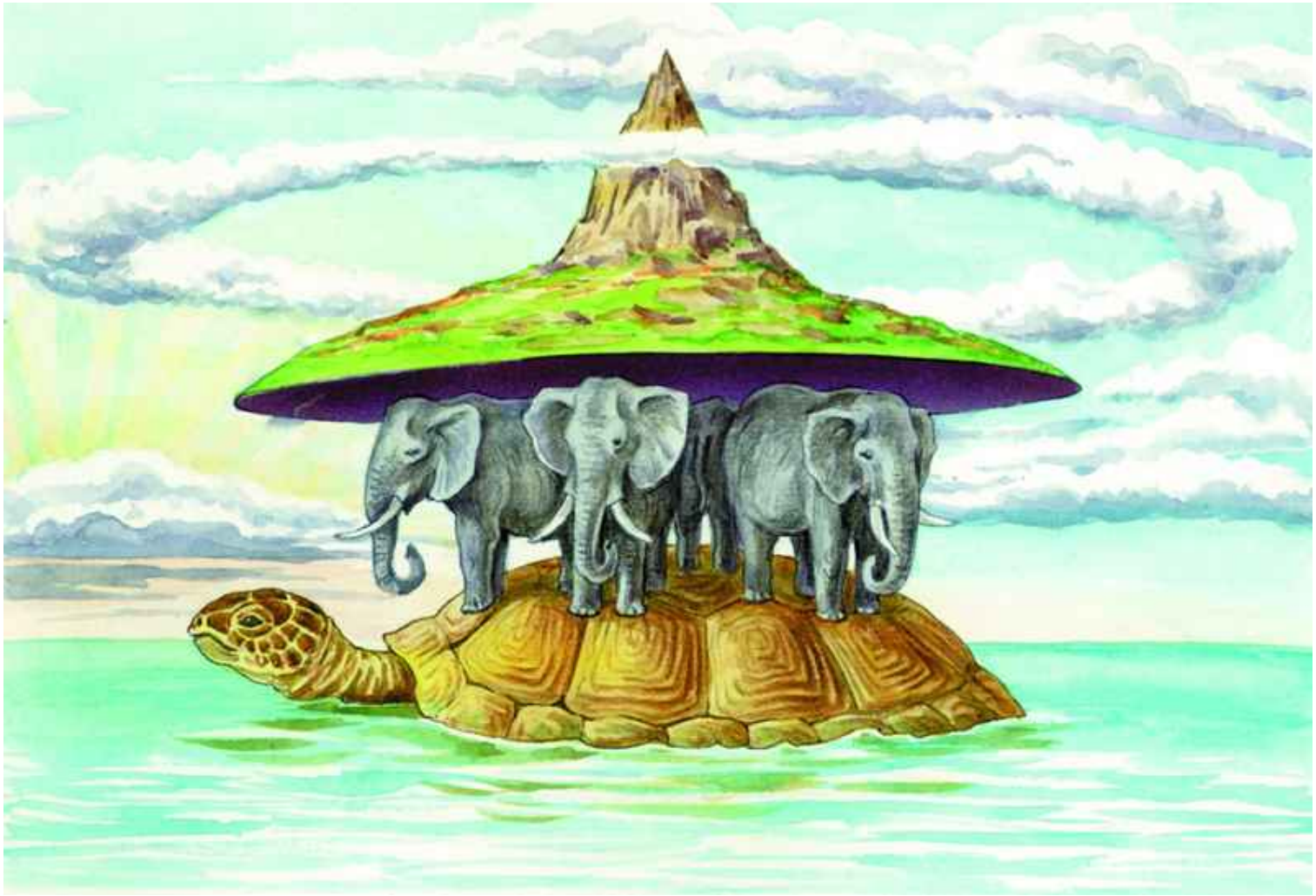
- Задание пакета для класса:
  `package ru.compscicenter.java2014;`
- Использование класса из пакета:
  - классы текущего пакета и пакета `java.lang` всегда видны
  - классы других пакетов доступны по полному имени с пакетом
  - можно использовать директиву `import`
- Имя пакета должно совпадать с именем директории:
  `ru/compscicenter/java2014/`

Импорт

- Импорт одного класса:
```
import ru.compscicenter.java2014.ExampleClass;
```
- Импорт всех классов пакета:
```
import ru.compscicenter.java2014.*;
```
- Импорт статических полей и методов:
```
import static java.lang.System.out;
import static java.util.Arrays.*;
```

Как работает импорт

- Директивы `import` позволяют компилятору получить полные имена всех используемых классов, полей и методов по их коротким именам
- В `class`-файл попадают полные имена, подстановка содержимого не происходит
- При запуске программы все используемые классы должны присутствовать в `classpath`



Объявление класса

```
package java.lang;

/**
 * The {@code Integer} class wraps a value of the primitive type
 * {@code int} in an object. An object of type {@code Integer}
 * contains a single field whose type is {@code int}.
 */
public class Integer {

    // class content

}
```

- здесь и далее примеры из JDK (с сокращениями)

Модификаторы доступа

- `public`
  доступ для всех
- `protected`
  доступ в пределах пакета и дочерних классов
- `private`
  доступ в пределах класса
- по умолчанию (нет ключевого слова)
  доступ в пределах пакета

Вложенные классы

- Можно объявить класс внутри другого класса
- Такие классы имеют доступ к `private`-членам друг друга
- Экземпляр вложенного класса связан с экземпляром внешнего класса
- Если связь не нужна, вложенный класс объявляют с модификатором `static`

Поля

```
package java.lang;

public class Integer {

    private final int value;

}
```

- Поля инициализируются значениями по умолчанию
- Модификатор `final` — значение должно быть присвоено ровно один раз к моменту завершения инициализации экземпляра

Конструкторы

```
package java.lang;

public class Integer {

    private final int value;

    public Integer(int value) {
        this.value = value;
    }
}
```

- Если не объявлен ни один конструктор, автоматически создается конструктор по умолчанию (без параметров)

Деструктор

- В Java нет деструкторов, сбор мусора автоматический
- Есть метод `void finalize()`, но пользоваться им не рекомендуется (не известно, когда будет вызван)
- При необходимости освободить ресурсы заводят обычный метод `void close()` или `void dispose()` и вызывают его явно

Методы

```
package java.lang;

public class Integer {

    private final int value;

    public int intValue() {
        return value;
    }
}
```

- Возможна перегрузка методов
  (несколько одноименных методов с разными параметрами)

Статические поля и методы

```
package java.lang;

public class Integer {

    public static final int MIN_VALUE = 0x80000000;

    public static int rotateRight(int i, int distance) {
        return (i >>> distance) | (i << -distance);
    }
}
```

- Статические поля и методы относятся не к экземпляру класса, а ко всему классу

Принцип «Tell, Don't Ask»

> Procedural code gets information then makes decisions.
> Object-oriented code tells objects to do things.
>
> *Alec Sharp*

- Правильно: говорить объектам, что вам от них нужно
- Неправильно: напрямую работать с внутренним состоянием объекта

Интерфейсы

- Интерфейс определяет контракт объекта, но не его реализацию
- Все `public`

```
package java.lang;

public interface CharSequence {

    int length();

    char charAt(int index);
}
```

Интерфейсы

- В Java 8 добавили default-методы
  (пример ниже придуман, такого метода на самом деле нет)

```
package java.lang;

public interface CharSequence {

    int length();

    CharSequence subSequence(int start, int end);

    default CharSequence subSequence(int start) {
        return subSequence(start, length());
    }
}
```

Абстрактные классы

- Нельзя создать экземпляр такого класса
- В отличие от интерфейса, в абстактном классе могут быть поля и не-public члены

```
package java.lang;

public abstract class Number {

    public abstract int intValue();
    public abstract long longValue();
    public abstract float floatValue();
    public abstract double doubleValue();
}
```

Перечисления

- Класс с фиксированным количеством экземпляров
- Может иметь поля и методы

```
package java.time;

public enum DayOfWeek {
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
}
```

Аннотации

- Метаданные к элементам программы
- Не могут содержать исполняемый код

```
package java.lang;

import java.lang.annotation.*;
import static java.lang.annotation.ElementType.*;

@Target({TYPE, FIELD, METHOD, PARAMETER,
         CONSTRUCTOR, LOCAL_VARIABLE})
@Retention(RetentionPolicy.SOURCE)
public @interface SuppressWarnings {
    String[] value();
}
```


НАСЛЕДОВАНИЕ
ПО ЗАКОНУ
И ЗАВЕЩАНИЮ
ПРАКТИЧЕСКОЕ ПОСОБИЕ
ТИХОМИРОВ

Объявление класса-наследника

```
package java.lang;

public final class StringBuilder
    extends AbstractStringBuilder {

    // derived class content

}
```

- Нет множественного наследования
- Все классы наследуют `java.lang.Object`

Объявление класса, реализующего интерфейс

```
package java.lang;

public final class String
    implements CharSequence {

    // implementation

}
```

- Класс может реализовывать сколько угодно интерфейсов

Модификатор final

- `final class MyClass {...}`
  нельзя создать класс-наследник
- `final void myMethod() {...}`
  нельзя переопределить метод в дочернем классе

Конструктор класса-наследника

```
package java.lang;

public final class StringBuilder
    extends AbstractStringBuilder {

    public StringBuilder() {
        super(16);
    }

    public StringBuilder(int capacity) {
        super(capacity);
    }
}
```

Переопределение методов

```
package java.lang;

public final class StringBuilder
    extends AbstractStringBuilder {

    @Override
    public StringBuilder append(String str) {
        super.append(str);
        return this;
    }
}
```

Оператор instanceof

- Позволяет проверить тип объекта в момент исполнения программы

```
Object obj = "hello world";
// obj instanceof Object          -> true
// obj instanceof String          -> true
// obj instanceof CharSequence    -> true
// obj instanceof Number          -> false
```

Liskov Substitution Principle

- Если S является подтипом T, тогда объекты типа T в программе могут быть замещены объектами типа S без каких-либо изменений желательных свойств этой программы
- Поведение наследуемых классов не должно противоречить поведению, заданному базовым классом, то есть поведение наследуемых классов должно быть ожидаемым для кода, использующего переменную базового типа.

Наследование и композиция

- Наследование — очень сильная связь
- Часто вместо наследования лучше использовать композицию (включение одного объекта в другой)

- Неправильно:
  - класс Train наследуется от List
  - класс Segment наследуется от Point

- Правильно:
  - класс Train содержит List
  - класс Segment содержит Point

Что сегодня узнали

- Что такое ООП
- Зачем классы раскладываются по пакетам
- Как в Java объявить класс, создать его экземпляры и работать с ними
- Как в Java реализуется инкапсуляция, наследование и полиморфизм