СПб АУ НОЦНТ РАН

Java 02

16.09.2015

Классы и интерфейсы. Сходства

```
abstract class A {
    abstract public void foo();
}

interface A {
    void foo();
}
```

- Если есть значение “A a”, можно написать “a.foo()”

Классы и интерфейсы. Сходства

```
abstract class A {
    abstract public void foo();
}

interface A {
    void foo();
}
```

- Если есть значение “A a”, можно написать “a.foo()”
- Нельзя написать “new A()”

Классы и интерфейсы. Сходства

```
abstract class A {
    abstract public void foo();
}

interface A {
    void foo();
}
```

- Если есть значение “A a”, можно написать “a.foo()”
- Нельзя написать “new A()”
- Отделили реализацию от абстракции и радуемся

Технические отличия

	Абстрактные классы	Интерфейсы
Модификаторы методов	public / protected	Неявно abstract public
Не abstract методы	Да	Нет
Состояние (поля)	Да	Нет
Множественное наследование	Нет	Да
Производительность	??	??

Технические отличия

	Абстрактные классы	Интерфейсы
Модификаторы методов	public / protected	Неявно abstract public
Не abstract методы	Да	Нет
Состояние (поля)	Да	Нет
Множественное наследование	Нет	Да
Производительность	??	??

Вывод

Технически классы более гибкие во всем кроме множественного наследования.

Сравнение с C++

- В C++ нет интерфейсов, как отдельной синтаксической конструкции

Сравнение с C++

- В C++ нет интерфейсов, как отдельной синтаксической конструкции
- Классы C++ технически умеют все тоже самое, что и классы в Java

Сравнение с C++

- В C++ нет интерфейсов, как отдельной синтаксической конструкции
- Классы C++ технически умеют все тоже самое, что и классы в Java
- *+ множественное наследование*

Сравнение с C++

- В C++ нет интерфейсов, как отдельной синтаксической конструкции
- Классы C++ технически умеют все тоже самое, что и классы в Java
- *+ множественное наследование*
- => Классы C++ можно использовать, как интерфейсы

Сравнение с C++

- В C++ нет интерфейсов, как отдельной синтаксической конструкции
- Классы C++ технически умеют все тоже самое, что и классы в Java
- *+ множественное наследование*
- => Классы C++ можно использовать, как интерфейсы
- **Минусы?** Множественное наследование с состоянием — очень сложная концепция (вспомним “class A : public virtual B”)
- Можно «примешивать» реализацию (mixins, java 8 default)

Концептуальные отличия

- Интерфейсы определяют поведение/контракт/признак, которым должен удовлетворять объект “x”

Концептуальные отличия

- Интерфейсы определяют поведение/контракт/признак, которым должен удовлетворять объект “x”
- Классы — в частности то же самое

Концептуальные отличия

- Интерфейсы определяют поведение/контракт/признак, которым должен удовлетворять объект “x”
- Классы — в частности то же самое
- Можно выделять общие части в базовые классы
- Классы — непересекающиеся множества

Примеры: LinkedList, List, BaseWindow, Drawable, HashSet, AbstractList, Named, Queue, CharSequence

Возможный ответ

- **Классы**: LinkedList (implements Queue, List), BaseWindow, HashSet, AbstractList
- **Интерфейсы**: Queue, Drawable, Named, CharSequence, List

Naming conventions:

- Классы: *Impl, Base*, Abstract*
- Интерфейсы: *ble, *ed

IO. Интерфейсы

- InputStream/OutputStream — работа с байтами
- Reader/Writer — работа с символами (чаще всего обертки над *Stream)

IO. Реализации

- Buffered* — оптимизация с помощью буферизации + readLine
- File* — работа с файлами
- Data*Stream — сериализация данных
- Scanner/PrintWriter/PrintStream — форматирование (nextInt, print(int))
- StringTokenizer — утилитный класс для токенизации строк

```
new BufferedInputStream(new FileInputStream("input.txt"));
new BufferedOutputStream(new FileOutputStream("output.txt"));
new Scanner(new File("input.txt"));
new PrintWriter("output.txt");
```