Интерфейсы

Интерфейсы в Java

Интерфейс - синтаксическая конструкция в коде программы, используемая для специфицирования услуг, предоставляемых классом или компонентом. Интерфейс определяет границу взаимодействия между классами или компонентами, специфицируя определенную абстракцию, которую осуществляет реализующая сторона.

Интерфейс в языке программирования Java - абстрактный тип, который используется для определения поведения, которое классы должны реализовать. Поведение интерфейсов в Java схоже с поведением абстрактных классов. Основным и главным отличием является то, что возможна реализация более одного интерфейса. Таким образом для интерфейсов разрешено множественное наследование(реализация).

Проблема отсутствия множественного наследования

Предположим у нас есть два сервиса по сохранению объектов. Первый сохраняет объект в текстовый файл. Второй сохраняет объект в базу данных. Логично что оба сервиса должны использовать полиморфный подход, т. е. метод принимающий в качестве параметра объект для сохранения должен использовать ссылку того или иного суперкласса. И есть класс Cat (описывающий кота) и нужно дать возможность сохранять этого кота и в текстовый файл и в базу данных одновременно.

Сервис для сохранения в текстовый файл

```java
public class TextFileSaveService {
	private File file;
	private String delimiter = ";";

	public TextFileSaveService(File file) {
		super();
		this.file = file;
	}
	public TextFileSaveService(File file, String delimiter) {
		super();
		this.file = file;
		this.delimiter = delimiter;
	}

	public void saveToCSVFile(GetDateToTextFile gdtt) throws IOException {
		String[] date = gdtt.textRepresentation().split(";");
		try (PrintWriter pw = new PrintWriter(file)) {
			for (int i = 0; i < date.length - 1; i++) {
				pw.print(date[i] + delimiter);
			}
			pw.print(date[date.length - 1]);
		} catch (IOException e) {
			throw e;
		}
	}
}
```

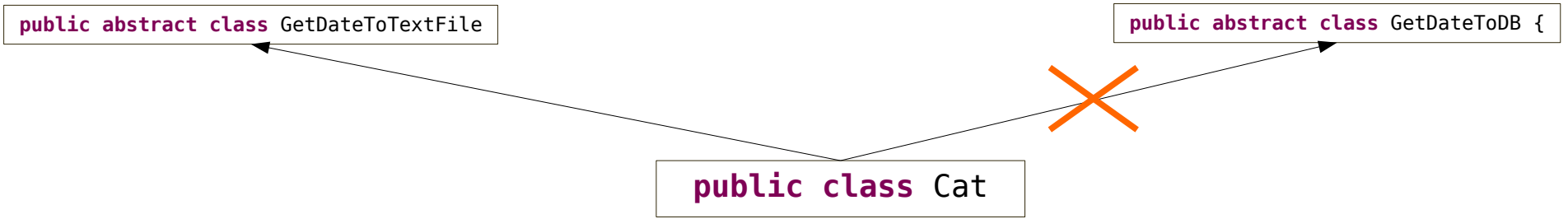
```java
public abstract class GetDateToTextFile {
	public abstract String textRepresentation();
}
```

Сервис для сохранения в базу данных

```java
public class DataBaseSaveService {

	public void saveToDB(GetDateToDB gdtdb) {
		String[] fields = gdtdb.getFielDescription();
		Object[] values = gdtdb.getFieldValue();
	}
}
```

```java
public abstract class GetDateToDB {
	public abstract String[] getFielDescription();

	public abstract Object[] getFieldValue();
}
```

Суть проблемы отсутствия множественного наследования




```
public class TextFileSaveService {
…..
       public void saveToCSVFile(GetDateToTextFile gdtt)
```

```
public class DataBaseSaveService {

       public void saveToDB(GetDateToDB gdtdb)
```

В Java множественное наследование запрещено, поэтому класс Cat не может реализовать два и более абстрактных класса и реализовать описанный функционал невозможно. Решением проблемы множественного наследования и стали интерфейсы.

Как объявлять интерфейс

Для объявления интерфейса используется ключевое слово **interface** после которого следует имя интерфейса. При объявлении интерфейса стоит помнить о некоторых особенностях.

В теле интерфейса вы можете объявлять:

- Абстрактные методы (ключевое слово abstract не обязательно)
- public методы с реализацией (ключевое слово default обязательно)
- private методы с реализацией
- Статические методы
- Открытые статические константы (все переменные в теле интерфейса автоматически)
- Вложенные интерфейсы
- Вложенные классы

В теле интерфейса запрещено объявлять:

- Поля
- Конструкторы

Как и в случае абстрактного класса, так и в случае интерфейса создать объект этого типа невозможно. Основная идея и упор делается именно на работу со ссылками.

Пример объявления интерфейсов

```
public interface GetDateToTextFile {

     public String textRepresentation();

}
```

Абстрактный метод

```
public interface GetDateToDB {
     public String[] getFielDescription();

     public Object[] getFieldValue();
}
```

Абстрактные методы

Как указать что класс реализует интерфейс

Для указания того, что класс реализует один и более интерфейсов после имени класса используется ключевое слово **implements** после чего следует список реализованных интерфейсов перечисленных через запятую. Внимание если не реализовать(переопределить) абстрактные методы интерфейса, то как и в случае абстрактного класса будет получена ошибка компиляции. Класс не реализующий все абстрактные методы интерфейса может быть только абстрактным.

Пример класс реализующего оба интерфейса

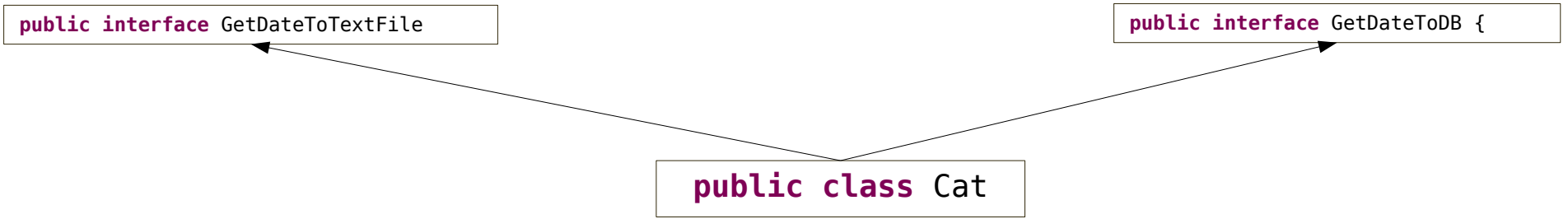
```
public class Cat implements GetDateToDB, GetDateToTextFile {
        private String name;

        public Cat(String name) {
                super();
                this.name = name;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        @Override
        public String textRepresentation() {
                return name;
        }

        @Override
        public String[] getFielDescription() {
                return new String[] { "String;name" };
        }

        @Override
        public Object[] getFieldValue() {
                return new Object[] { name };
        }

        @Override
        public String toString() {
                return "Cat [name=" + name + "]";
        }
}
```

Класс реализует два интерфейса

Реализация абстрактного метода интерфейса GetDateToTextFile

Реализация абстрактных методов интерфейса GetDateToDB

Множественное наследование с использованием интерфейсов




```
public class TextFileSaveService {
…..
       public void saveToCSVFile(GetDateToTextFile gdtt)
```

```
public class DataBaseSaveService {

       public void saveToDB(GetDateToDB gdtdb)
```

В Java класс может реализовать более одного интерфейса, поэтому класс Cat может реализовать оба интерфейса. И как следствие объекты этого класса могут использоваться в обоих сервисах.

Пример использования

```
public class Main {

     public static void main(String[] args) {

          Cat cat = new Cat("Vaska");

          TextFileSaveService ts = new TextFileSaveService(new File("cat.txt"));

          try {
               ts.saveToCSVFile(cat);
          } catch (IOException e) {
               e.printStackTrace();
          }

          DataBaseSaveService db = new DataBaseSaveService();

          db.saveToDB(cat);

     }

}
```

Использование ссылки класса Cat

Использование ссылки класса Cat

Для интерфейсов работает механизм наследования

```
public class Main {

     public static void main(String[] args) {

          Cat cat = new Cat("Vaska");

          GetDateToTextFile gt = cat;
          GetDateToDB gdb = cat;

     }
}
```

Восходящее преобразование типов

Использование интерфейсов базируется на механизме наследования, поэтому восходящее преобразование типов работает и в этом случае. Ссылке типа интерфейс вы можете присвоить значение ссылки типа класс его реализующий. Это также реализует полиморфизм подтипов. В любой точке программы где может быть использована ссылка типа интерфейс вы можете использовать ссылку на класс его реализующий. Как это было сделано в реализации обоих сервисов.

К чему есть доступ по ссылке типа интерфейс

```
public class Main {

     public static void main(String[] args) {

          Cat cat = new Cat("Vaska");

          GetDateToTextFile gt = cat;
          GetDateToDB gdb = cat;

          gt.textRepresentation();

          gdb.getFielDescription();

     }
}
```

◄ Восходящее преобразование типов

◄ Доступ определяется типом ссылки

При восходящем преобразовании к ссылке типа интерфейс применяются те же правила, что и при наследовании. Т.е. доступ есть только к тому, что доступно из типа ссылки (абстрактные, и public методы с реализацией) указанных интерфейсов.

Вопросы и ответы к ним

Существует ли ограничение на количество реализуемых интерфейсов?

Да, согласно спецификации JVM [2] и формата class файла[3] класс может реализовать максимум 65535 интерфейсов. Однако это число столько велико, что вряд ли когда либо понадобиться одному классу реализовывать большее количество интерфейсов.

Как объявлять класс при использовании наследования и реализации интерфейсов одновременно?

В этом и последующих случаях стоит запомнить простое правило (наследование важнее реализации). Т.е порядок объявления таков

```
class ClassName extends SuperClass implements InterfaceA, interfaceB...
```

Схожесть и различие поведения интерфейсов и абстрактных классов

Схожесть

- Невозможно создать объект.
- Использование механизма восходящего преобразования.
- Возможность объявлять статические методы.
- Возможность объявлять абстрактные методы.
- Возможность объявления методов с реализацией (в интерфейсах default и private).
- Возможность описания вложенных классов и интерфейсов.

Различие

- Для интерфейсов разрешено множественное наследование(реализация) для абстрактных классов нет.
- В абстрактных классах могут быть описаны поля, в интерфейсах нет.
- В абстрактных классах могут быть описаны конструкторы, в интерфейсах нет.

Объявление статических констант в интерфейсе

В интерфейсах вы можете объявить статические public константы (указание модификаторов final и static не обязательно, они будут добавлены автоматически).

```
public interface SampleInterface {

    public String ID = "75234";

}
```

Статическая константа

```
public class Main {

    public static void main(String[] args) {

        System.out.println(SampleInterface.ID);

    }

}
```

Использование константы

Объявление статических методов в интерфейсе

В интерфейсах вы можете объявить статические public методы.

```
public interface SampleInterface {

    public static void printHello(String name) {
        System.out.println("Hello " + name);

    }
}
```

Статический метод с реализацией

```
public class Main {

    public static void main(String[] args) {
        String name = "Alex";
        SampleInterface.printHello(name);
    }

}
```

Использование статического метода интерфейса

Методы по умолчанию

Методы по умолчанию (default methods) — не статические методы с реализацией, объявленные в теле интерфейса. Впервые появились в Java 1.8. Для объявления используется ключевое слово default. Если класс реализует интерфейс, он может, но не обязан, реализовать методы, реализованные в интерфейсе. Класс наследует реализацию по умолчанию.

Пример интерфейса с методом по умолчанию

```
public interface SampleInterface {

    public default String getMessage() {
        return "Default message";
    }
}
```

Метод по умолчанию

Метод по умолчанию объявлен с помощью ключевого слова **default**.

Класс, реализующий интерфейс

```
public class ClassA implements SampleInterface {
    private String message;

    public ClassA(String message) {
        super();
        this.message = message;
    }

    @Override
    public String getMessage() {
        return message;
    }

    @Override
    public String toString() {
        return "ClassA [message=" + message + "]";
    }

}
```

Реализация интерфейса

Переопределение метода по умолчанию

Класс, реализующий интерфейс

```
public class ClassB implements SampleInterface {
    private String message;

    public ClassB(String message) {
        super();
        this.message = message;
    }

    @Override
    public String toString() {
        return "ClassB [message=" + message + "]";
    }

}
```

Реализация интерфейса

Работа с описанными классами

```
public class Main {

    public static void main(String[] args) {

        SampleInterface a = new ClassA("Hello");

        SampleInterface b = new ClassB("Hello");

        System.out.println(a.getMessage());
        System.out.println(b.getMessage());
    }

}
```

ClassA реализует указанный интерфейс и переопределяет default метод, ClassB реализует интерфейс, но не переопределяет default метод. При вызове этого метода в первом случае был вызван переопределенный метод в классе, во втором реализация в интерфейсе.

Описание вложенных интерфейсов

В теле интерфейса вы можете объявить классы и интерфейсы. И первые и вторые автоматически получают модификатор static.

```
public interface TopInterface {

    public interface NestedInterface {
        public String getTextRepresentation();
    }

}
```

Вложенный интерфейс

```
public class Cat implements TopInterface.NestedInterface {

    @Override
    public String getTextRepresentation() {
        return "I am a cat";
    }
}
```

Его реализация

Наследование интерфейсов

Для интерфейсов также реализован механизм наследования. Один интерфейс может наследовать несколько других интерфейсов (для интерфейсов наследование множественное). Поэтому существует такое понятие как суперинтерфейс и подинтерфейс соответственно.

```
interface InterfaceA {
    public String getId();
}

interface InterfaceB{
    public String getName();
}

interface InterfaceC extends InterfaceA,InterfaceB{
    public String getVoice();
}
```

Наследование интерфейсов

Особенности реализации

При реализации подинтерфейса нужно будет реализовать все методы его суперинтерфейсов и методы описанные непосредственно в нем.

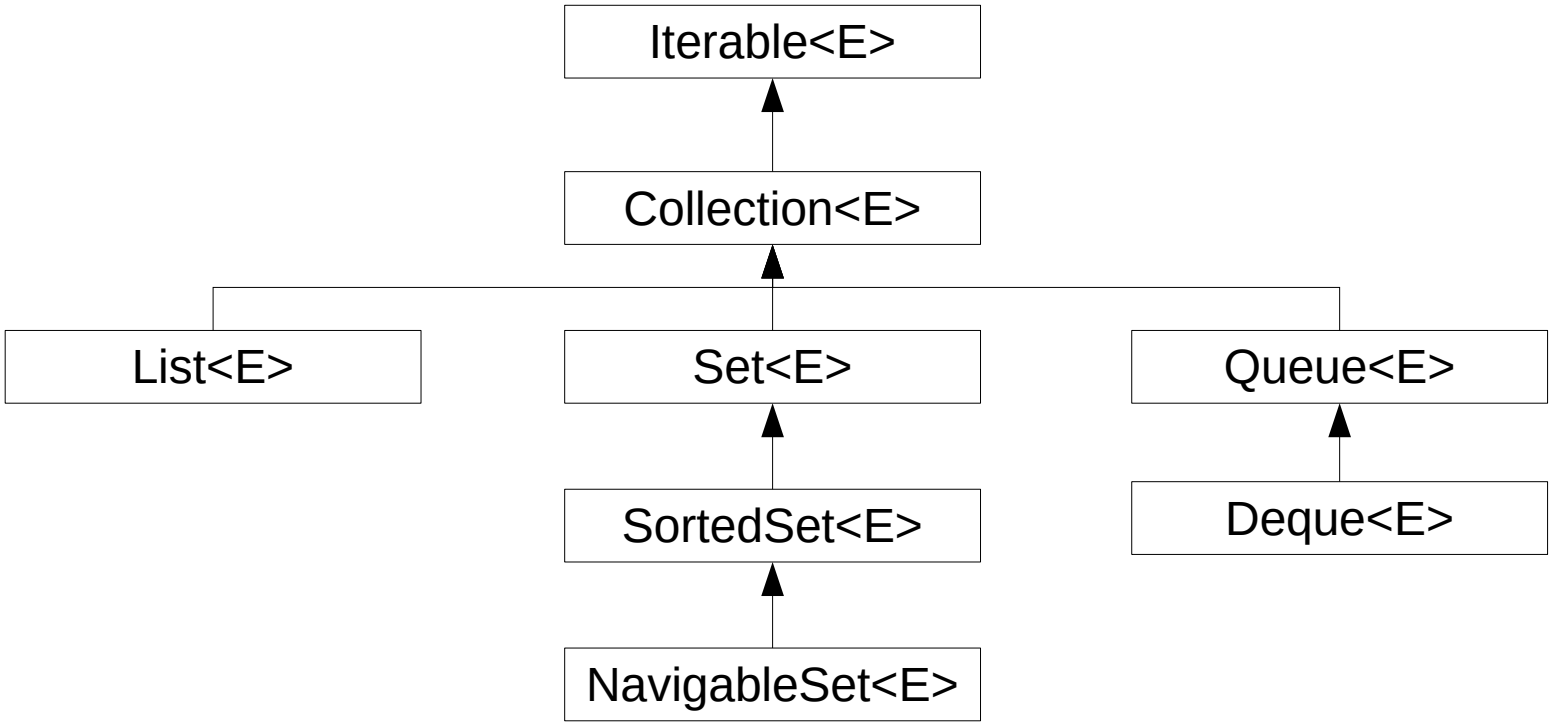
```
public class Cat implements InterfaceC{

    @Override
    public String getName() {
        return "Vaska";
    }

    @Override
    public String getId() {
        return "Paws and tail my id ";
    }

    @Override
    public String getVoice() {
        return "Meow";
    }
}
```

Реализация метода интерфейса InterfaceB

Реализация метода интерфейса InterfaceA

Реализация метода интерфейса InterfaceC

Пример использования наследования интерфейсов в стандартной библиотеке



Хорошим примером использования иерархий интерфейсов является иерархия Collection в Java.

Использование интерфейсов в стандартной библиотеке

Помимо использования интерфейсов при проектировании архитектуры приложения, они активно используются в стандартной библиотеке. Это позволяет сделать методы универсальными относительно данных с которыми они работают. В качестве примера приведем способ сортировки массива пользовательских типов данных.

Класс Cat

```java
public class Cat {
	private String name;
	private int age;

	public Cat(String name, int age) {
		super();
		this.name = name;
		this.age = age;
	}

	public Cat() {
		super();
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	@Override
	public String toString() {
		return "Cat [name=" + name + ", age=" + age + "]";
	}

}
```

Задача сортировки массива ссылок типа Cat

```
public class Main {

    public static void main(String[] args) {

        Cat cat1 = new Cat("Vaska", 6);
        Cat cat2 = new Cat("Luska", 2);
        Cat cat3 = new Cat("Umka", 4);
        Cat cat4 = new Cat("Barsic", 8);

        Cat[] cats = new Cat[] { cat1, cat2, cat3, cat4 };

    }
}
```

Для сортировки массива ссылок типа Cat, можно использовать метод Arrays.sort с двумя параметрами. Первым выступает ссылка на сортируемый массив, а вторым параметром реализация интерфейса Comparator для объектов сортируемого массива.

Интерфейс Comparator

В Java существует интерфейс java.util.Comparator. Этот интерфейс используется для указания отношения полного порядка (более подробно будет рассмотрен в последующих лекциях).

Абстрактный метод интерфейса Comparator

В качестве абстрактного метода интерфейса Comparator выступает метод **int compare(Object o1, Object o2)**. Его реализация должна вернуть положительное число (любое) если o1 по выбранному вами критерию больше o2, отрицательное число если o1 меньше o2 и 0 в случае их равенства. Пока будем использовать не обобщенную версию данного интерфейса.

Для корректной реализации стоит определить по какому критерию вы будете сравнивать объекты сортируемого массива. После чего реализовать этот интерфейс.

Пример реализации Comparator

```
import java.util.Comparator;

public class CatAgeComparator implements Comparator {

     @Override
     public int compare(Object o1, Object o2) {
          Cat cat1 = (Cat) o1;
          Cat cat2 = (Cat) o2;

          if (cat1.getAge() > cat2.getAge()) {
               return 1;
          }
          if (cat1.getAge() < cat2.getAge()) {
               return -1;
          }
          return 0;
     }
}
```

Импорт

Приведение к нужному типу ссылки

Сравнение по выбранному критерию

В качестве критерия для сравнения был выбран возраст кота. Сортировать будем по возрастанию.

Использование реализации Comparator

```
import java.util.Arrays;

public class Main {

     public static void main(String[] args) {

          Cat cat1 = new Cat("Vaska", 6);
          Cat cat2 = new Cat("Luska", 2);
          Cat cat3 = new Cat("Umka", 4);
          Cat cat4 = new Cat("Barsic", 8);

          Cat[] cats = new Cat[] { cat1, cat2, cat3, cat4 };


                    Использование реализации Comparator


          Arrays.sort(cats, new CatAgeComparator());

          for (int i = 0; i < cats.length; i++) {
               System.out.println(cats[i]);
          }
     }
}
```

Сортировка массива с наличием null

Для того, что бы отсортировать массив в котором есть null можно использовать статические методы Comparator.nullsFirst и Comparator.nullsLast которому в качестве параметра нужно передать вашу реализацию Comparator.

```
public class Main {

	public static void main(String[] args) {

		Cat cat1 = new Cat("Vaska", 6);
		Cat cat2 = new Cat("Luska", 2);
		Cat cat3 = new Cat("Umka", 4);
		Cat cat4 = new Cat("Barsic", 8);

		Cat[] cats = new Cat[] { cat1, cat2, null, cat3, cat4 };

		Arrays.sort(cats, Comparator.nullsFirst(new CatAgeComparator()));

		for (int i = 0; i < cats.length; i++) {
			System.out.println(cats[i]);
		}
	}
}
```

Сортируем с null в массиве

Задание для самостоятельной проработки. Основной уровень.

1) Дополните реализацию группы Студентов (задание прошлой лекции) возможностью сортировки массива студентов по фамилии. Для этого в класс Группа добавьте метод sortStudentsByLastName().
2) Создайте отдельный класс который реализует считывание характеристик студента с клавиатуры (имя, фамилии и т. д.). Создание и возврат студента на основе считанных данных. Используете методы этого класса для считывания и добавления студента в группу.

Задание для самостоятельной проработки. Продвинутый уровень.

1) Объявите интерфейс

```
public interface StringConverter{
    public String toStringRepresentation (Student student);
    public Student fromStringRepresentation (String str);
}
```

2) Объявите класс CSVStringConverter реализующий указанный интерфейс. Логика реализации следующая — на основе Студента создать строку с его CSV представлением и наоборот на основе этой строки создать Студента. Ссылка на объяснение формата CSV - https://ru.wikipedia.org/wiki/CSV .

Список литературы

1) Герберт Шилдт Java 8. Полное руководство 9-е издание ISBN 978-5-8459-1918-2
2) https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.11
3) https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.1