Java Language

---

1. Пакеты (package): правила создания, правила именования, назначение.

Пакеты – это контейнеры классов, которые используются для разделения пространства имен классов и позволяют логически объединить классы в наборы. Основные классы java входят в пакет java.lang. Различные вспомогательные классы располагаются в пакете в java.util.

Структура пакетов в точности отображает структуру файловой системы. Все файлы с исходными кодами и байт-кодами, образующие один пакет, хранятся в одном каталоге файловой системы. Пакет может содержать подпакеты.

Наименование пакета может быть любым, но необходимо соблюдать его уникальность в проекте. Компоненты доменного имени в объявлении package перечисляются в обратном порядке: package com.google.android.maps.

Все имена классов и интерфейсов в пакете должны быть уникальными. Имена классов в разных пакетах могут совпадать. Чтобы указать, что класс принадлежит определенному пакету, следует использовать директиву package, после которой указывается наименование (путь) пакета:

```
package company.common;

public class HelloWorld{
    public static void main(String[] args){
        System.out.println ("Hello, World!");
    }
}
```

Если оператор package не указан, классы попадают в безымянное пространство имен, используемое по умолчанию.

Пакеты регулируют права доступа к классам и подклассам. Если ни один модификатор доступа не указан, то класс, метод или переменная является доступной всем методам в том же самом пакете.

Чтобы использовать какой-то класс в коде другого класса, необходимо его импортировать, написав до объявления класса строку: import company.common.HelloWorld. Можно импортировать весь пакет: import company.common.*

---

2. Статические методы, особенности работы со статическими методами

Статические методы являются методами класса, не привязаны ни к какому объекту и не содержат указателя this на конкретный экземпляр, вызвавший метод. Статические методы реализуют парадигму «раннего связывания», жестко определяющую версию метода на этапе компиляции. По причине недоступности указателя this статические методы не могут обращаться к нестатическим полям и методам напрямую, так как они не «знают», к какому объекту относятся, да и сам экземпляр класса может быть не создан.

Для объявления статических методов перед их объявлением используется ключевое слово static. Вызов статического метода всегда следует осуществлять с помощью указания на имя класса, а не объекта. Переопределение статических методов невозможно, так как статические методы связываются во время компиляции, а не в runtime, то есть полиморфизм на статические методы не распространяется.

Статические методы можно импортировать при помощи ключевых слов import static, после чего к статическим методам можно будет обращаться без указания имени класса:

```
import static java.lang.System.out;
public class StaticImport {
```

```
	public static void main(String[] args) {
		out.println("hi");
	}
}
```

Статические методы следует использовать, когда не нужен доступ к состоянию объекта, а все параметры задаются явно, или когда методу нужен доступ только к статическим полям класса.

---

3. Анонимные классы: объявление и правила работы

Анонимный класс - это локальный класс без имени. Анонимные или безымянные классы декларируются внутри методов основного класса и могут быть использованы только внутри этих методов. Анонимный класс расширяет другой класс, применяется для придания уникальной функциональности отдельно взятому экземпляру. То есть можно объявить анонимный класс, который будет расширять другой класс или реализовывать интерфейс при объявлении единственного объекта, когда остальным объектам этого класса будет соответствовать реализация метода, определенная в самом классе.

Основное требование к анонимным классами - они должны наследовать или расширять уже существующий класс или интерфейс. Анонимные классы не могут содержать статические поля и методы, кроме final static.

Классический пример использования анонимного класса - запуск потоков:

```
new Thread(new Runnable() {
	public void run() {
		//какой-то код
   }
}).start();
```

Синтаксис создания анонимного класса базируется на использовании оператора new с именем класса (интерфейса) и телом анонимного класса. Невозможно определить или переопределись конструктор анонимного класса. Анонимные классы могут быть вложены друг в друга, хотя так делать не рекомендуется из-за сложность понимания такого кода.

Ситуации, в которых следует использовать внутренние классы:

- выделение самостоятельной логической части сложного класса;
- сокрытие реализации;
- одномоментное использование переопределенных методов. В том числе обработка событий;
- запуск потоков выполнения.

---

4. Вложенные классы: объявление и правила работы

Классы могут взаимодействовать друг с другом путем организации структуры с определением одного класса в теле другого. Это делает код более эффективным и понятным, а также позволяет сокрыть реализацию, так как внутренний класс может быть не виден вне основного класса. Вложенные классы могут быть статическими и нестатическими. Статические классы могут обращаться к членам включающего класса только через его объект. А нестатические классы имеют доступ ко всем членам включающего класса.

Нестатические вложенные классы называются внутренними (inner) классами. Доступ к элементам внутреннего класса из внешнего возможен только через объект внутреннего класса, который создается в методе внешнего класса.

```
public class Ship {
```

```
	private class Engine {
		public void launch() {
			System.out.println("Start engine");
		}
	}
	public void initShip() {
		Engine engine = new Engine();
		engine.launch();
	}
}
```

Методы внутреннего класса имеют доступ ко всем полям и методам внешнего класса, а внешний класс может получить доступ к содержимому внутреннего класса только после создания объекта внутренного класса, при этом доступ будет разрешен и к private полям и методам.

Внутренние классы могут быть объявлены как final, abstract, private, protected, public. Объявление внутреннего класса как private обеспечивает полную недосягаемость вне внешнего класса. Внутренние классы могут быть объявлены внутри методов или логических блоков внешнего класса. В этом случае видимость внутреннего класса определяется видимостью того блока, где он объявлен.

Внутренние классы могут наследовать другие классы, могут реализовывать интерфейсы, а так же сами могут быть базовыми классами для других классов. То есть этот механизм может решить проблему множественного наследования.

Внутренние классы не могут содержать статические поля и методы, кроме final static, но могут наследовать.

Статические вложенные классы называются вложенными (nested) классами. Вложенный класс логически связан с внешним классом, однако может быть использован независимо от него. Такие классы объявляются с ключевым словом static. Если класс вложен в интерфейс, то он является статическим по умолчанию.

Для доступа к нестатическим полям и методам внешнего класса, статический вложенный класс должен создать объект внешнего класса, а к статическим полям и методам он имеет доступ напрямую. Из этого следует, что для создания объекта статического вложенного класса нет необходимости создавать объект внешнего класса.

Вложенные классы могут наследовать другие классы, могут реализовывать интерфейсы, а так же сами могут быть базовыми классами для других классов. Производный класса вложенного класса теряет доступ к членам внешнего класса при наследовании.

Статические методы вложенного класса при вызове требуют указания полного пути к нему. То есть, если внешний класс это Car, а вложенный статический класс это Driver, в котором есть статический метод park(), то для вызова требуется такая строка:

```
Car.Driver.park();
```

Безымянные вложенные классы, которые объявляются при помощи оператора new называются анонимными классами. Применяются для придания уникальной функциональности отдельно взятому экземпляру. То есть можно объявить анонимный класс, который будет расширять другой класс или реализовывать интерфейс при объявлении единственного объекта, когда остальным объектам этого класса будет соответствовать реализация метода, определенная в самом классе. Основное требование к анонимным классами - они должны наследовать или расширять уже существующий класс или интерфейс.

Анонимные классы как правило используются для переопределения нескольких методов и/или создания собственных методов объекта, при этом в создании нового полноценного класса нет необходимости из-за узкой направленности или одноразового применения.

Невозможно определить и переопределить конструктор анонимного класса. Анонимные классы не могут содержать статические поля и методы, кроме final static, но могут наследовать. Допускают вложенность, однако это не рекомендуется из-за усложнения читаемости кода.

Локальные классы.
Объявляются и могут быть использованы только внутри методов внешнего класса. Имеют доступ к членам внешнего класса. Имеют доступ как к локальным переменным, так и к параметрам метода при одном условии - переменные и параметры, используемые локальным классом, должны быть объявлены final. Локальные классы не могут содержать статические поля и методы, кроме final static, но могут наследовать.

---

5. Интерфейсы в Java 7: объявление, методы и поля интерфейсов.

Назначение интерфейса — описание или спецификация функциональности, которую должен реализовывать каждый класс, его имплементирующий. Класс, реализующий интерфейс, предоставляет к использованию объявленный интерфейс в виде набора public-методов в полном объеме. Интерфейсы подобны полностью абстрактным классам, хотя и не являются классами.

В интерфейсе не могут быть объявлены поля без инициализации, все поля неявно public final static. Внутри интерфейса не может быть реализован ни один из объявленных методов. Все объявленные в интерфейсе методы неявно public и abstract.

```
public interface BankAction {
	int startCash = 0;
	void openAccount();
	void blockAccount();
	void closeAccount();
}

public class TinkoffBankAction implements BankAction {
	private void openAccount() {
		System.out.println("Account is open");
	}

	private void blockAccount() {
		System.out.println("Account is blocked");
	}

	private void closeAccount() {
		System.out.println("Account is closed");
	}
}
```

Класс может реализовывать любое число интерфейсов, которые указываются через запятую в объявлении класса после слова implements. Такой класс обязан предоставить реализацию всех методов, которые определены в интерфейсе, или же объявить себя абстрактным. Иными словами, интерфейс - это указание на то, что должен делать класс без указания на то, как именно это делать.

Можно объявить ссылку типа интерфейса, она сможет указывать на объект любого класса, который реализует этот интерфейс. При вызове метода через эту ссылку будет вызываться его реализованная версия метода у объекта, на который указывает эта ссылка, используя механизм динамического связывания.

6. Перегрузка методов: правила перегрузки, разрешение перегрузки

Java разрешает определение внутри одного класса двух или более методов с одним именем, если списки их параметров различны по количеству и/или типу. Такие методы называются перегруженными. Возвращаемые типы перегруженных методов могут быть различными, но если методы различаются только типом возвращаемого значения, этого недостаточно для перегрузки метода.

Перегрузка реализует «раннее связывание», то есть версия вызываемого метода определяется на этапе компиляции. Методы с одинаковыми именами, но с различными списком параметров и возвращаемыми значениями могут находиться в разных классах одной цепочки наследования и также будут перегруженными. Статические методы могут перегружаться нестатическими, и наоборот.

Механизм перегрузки снижает гибкость кода, однако позволяет избежать ошибок при обращении к перегруженным методам, которые отслеживаются на этапе компиляции. Следует избегать ситуаций, когда компилятор будет не в состоянии выбрать правильный метод, например, из-за неявного приведения типов. Следует также по возможности избегать перегрузку с одинаковым числом параметров, так как это может привести с ошибкам в связи с тем же неявным приведением типов компилятором.

7. Абстрактные классы и абстрактные методы

Абстрактные классы объявляются с ключевым словом abstract и содержат объявления абстрактных методов, которые не реализованы в этих классах, а будут реализованы в подклассах. Объекты таких классов создать нельзя с помощью оператора new, но можно создать объекты подклассов, которые реализуют все эти методы. При этом допустимо объявлять ссылку на абстрактный класс, но инициализировать ее можно только объектом производного от него класса. Абстрактные классы могут содержать и полностью реализованные методы, а также конструкторы и поля данных.

```
abstract class AnyClass {
	int someField;
	void doAction();
	void print() {
		System.out.println(someField);
	}
}

public class Main {
	public static void main(String[ ] args) {
		AnyClass anyField;
		//anyField = new AnyClass();		нельзя создать объект!
		anyField = new AnySubClass();		//создаем объект подкласса абстрактного
класса
		anyField.print();
	}
}
```

8. Статические поля, статические константные поля.

Поле данных, объявленное в классе как static, является общим для всех объектов класса. Может быть использовано без создания экземпляра класса. Если один объект изменит значение такого поля, то это изменение увидят все объекты. Для объявления статических переменных перед их объявлением используется ключевое слово static.

Константное поле - это поле, значение которого не изменяется в процессе работы программы. Константами принято называть public static final поля, которые сразу проинциализированы. Согласно Code Convention, имена констант должны содержать только большие буквы, разделенные нижним подчеркиванием:

```
public static final FIELD_SIZE = 5;
```

Статические переменные инициализируются во время загрузки класса. К статическим переменным нужно обращаться через имя класса. Локальные переменные, как и абстрактные методы, не могут быть объявлены как static.

---

9. Интерфейсы в Java 8

Назначение интерфейса — описание или спецификация функциональности, которую должен реализовывать каждый класс, его имплементирующий. Класс, реализующий интерфейс, предоставляет к использованию объявленный интерфейс в виде набора public-методов в полном объеме. Интерфейсы подобны полностью абстрактным классам, хотя и не являются классами.

В интерфейсе не могут быть объявлены поля без инициализации, все поля неявно public final static. Внутри интерфейса не может быть реализован ни один из объявленных методов. Все объявленные в интерфейсе методы неявно public и abstract.

```
public interface BankAction {
	int startCash = 0;
	void openAccount();
	void blockAccount();
	void closeAccount();
}

public class TinkoffBankAction implements BankAction {
	private void openAccount() {
		System.out.println("Account is open");
	}

	private void blockAccount() {
		System.out.println("Account is blocked");
	}

	private void closeAccount() {
		System.out.println("Account is closed");
	}
}
```

Класс может реализовывать любое число интерфейсов, которые указываются через запятую в объявлении класса после слова implements. Такой класс обязан предоставить реализацию всех методов, которые определены в интерфейсе, или же объявить себя абстрактным. Иными словами, интерфейс - это указание на то, что должен делать класс без указания на то, как именно это делать.

Можно объявить ссылку типа интерфейса, она сможет указывать на объект любого класса, который реализует этот интерфейс. При вызове метода через эту ссылку будет вызываться его реализованная версия метода у объекта, на который указывает эта ссылка, используя механизм динамического связывания.

Начиная с Java 8 появилась возможность использовать в интерфейсах дефолтные и статические методы.

Default-методы - это неабстрактные реализации методов. Классы, которые реализуют интерфейс с default-методами, обязаны переопределять только абстрактные методы, а переопределение default-методов необязательно. То есть классы, которые реализуют интерфейс с default-методами, могут пользоваться уже реализованными методами. Default-методы следует использоваться с осторожностью, чтобы не столкнуться с проблемой ромба множественного наследования.

Static-методы - это статические неабстрактные методы, они являются частью интерфейса и не могут быть использованы в классах, реализующих такой интерфейс. Они не могут быть предопределены - если в классе есть метод с такой же сигнатурой, он будет являться самостоятельным методом. Используются как вспомогательные методы для default-методов.

Функциональный интерфейс - это интерфейс с только одним абстрактным методом, для обозначения такого интерфейса используется аннотация @FunctionalInterface. Аннотация не является обязательной, но предостерегает программиста от случайного добавления абстрактных методов в функциональный интерфейс. Функциональные интерфейсы могут содержать сколько угодно default и static методов, но абстрактный метод должен быть только один. Функциональные интерфейсы позволяют использовать лямбда-выражения для создания анонимных классов, реализующих этот интерфейс.