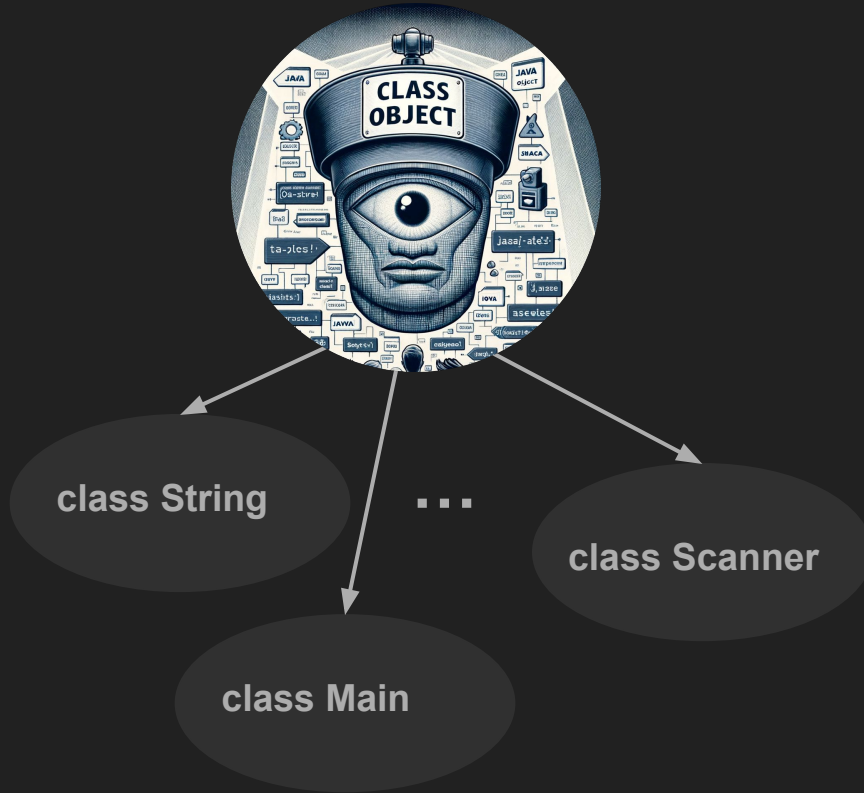




Object & String



Великая ложь



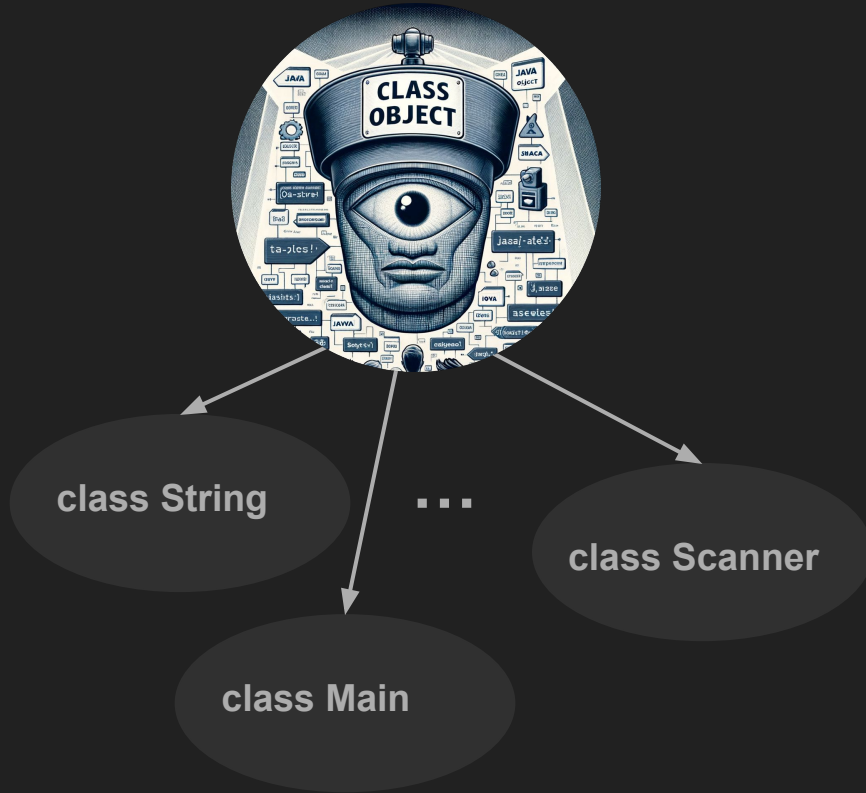
Класс Object в Java - это родитель всех классов.

Представьте, что в мире Java все классы - это дети, а класс Object - это их общий родитель.

Это значит, что любой класс в Java автоматически унаследует некоторые базовые свойства и методы от класса Object, даже если мы этого специально не указываем.



Великая ложь

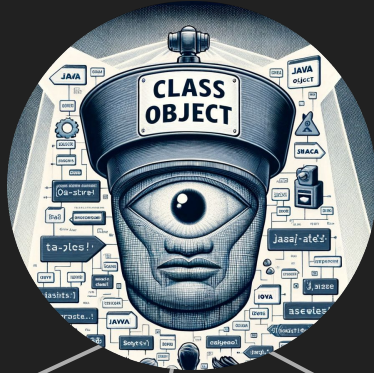


Некоторые такие унаследованные от Object вещи:

- `equals(Object obj)`: Этот метод помогает сказать, одинаковые ли два объекта. Например, если у вас два яблока, и вы хотите узнать, одно и то же это яблоко или два разных, `equals` поможет в этом.
- `toString()`: Этот метод помогает "представить" объект в виде строки. Если бы объект мог говорить, он бы сказал: "Вот как я выгляжу в виде текста!".
- `hashCode()`: Это как уникальный код объекта, похожий на номер паспорта для человека. Он помогает идентифицировать объект, особенно когда их много, и нужно быстро найти один из них.
- `getClass()`: Этот метод говорит, к какому "классу" относится объект. Это как спросить у кого-то: "Кто ты по профессии?".



Великая ложь



Класс Object в Java - это основа для всех классов.

Он предоставляет некоторые базовые "инструменты" и "документы", которые помогают объектам "жить" в мире Java, общаться друг с другом и быть уникальными.

class String

...

class Scanner

class Main



Сравнение строк

```
package l24;

public class StringComparing {
    public static void main(String[] args) {
        String str1 = "hello";
        String str2 = "hello";
        String str3 = new String("hello");

        // true потому что переменные
        // указывают на один и тот же объект
        System.out.println(str1 == str2);
        // false потому что переменные
        // указывают на разные объекты
        System.out.println(str1 == str3);
        // true потому что сравниваем
        // значения строк
        System.out.println(str1.equals(str3));
    }
}
```

Оператор сравнения `==` сравнивает “непосредственные значения” наших переменных.

В случае с ссылочными типами данных (все создающиеся через класс, то есть все не примитивные) оператор `==` сравнивает только ссылки на данные, а не сами данные.

Чтобы сравнивать сами данные, то есть значения, в случае с ссылочными типами данных следует использовать метод `equals`, который доступен во всех классах (наследуется от класса `Object`).



Сравнение строк

```
package l24;

public class StringComparing {
    public static void main(String[] args) {
        String str1 = "hello";
        String str2 = "hello";
        String str3 = new String("hello");

        // true потому что переменные
        // указывают на один и тот же объект
        System.out.println(str1 == str2);
        // false потому что переменные
        // указывают на разные объекты
        System.out.println(str1 == str3);
        // true потому что сравниваем
        // значения строк
        System.out.println(str1.equals(str3));
    }
}
```

String в Java - это неизменяемый (immutable) класс.

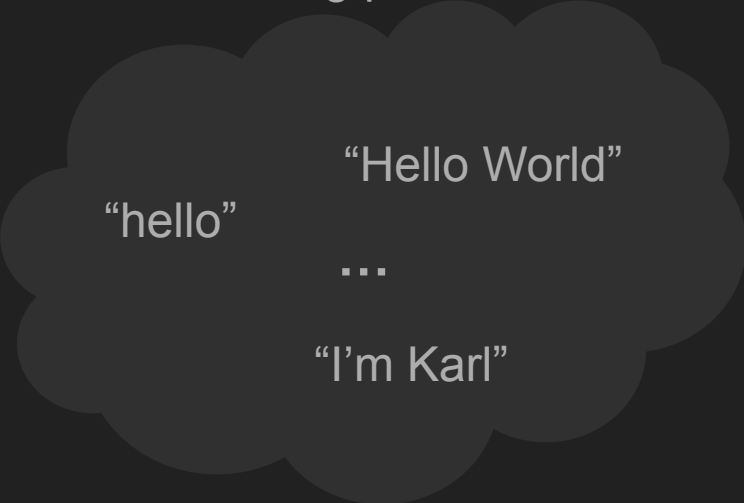
Это означает, что однажды созданный объект String не может быть изменен.

Если кажется, что строка изменяется (например, при конкатенации), на самом деле создается новый объект String.



String pool

String pool:



"hello"
"Hello World"
...
"I'm Karl"

String Pool - это специальная область в памяти, предназначенная для хранения уникальных строковых литералов в Java.

Это позволяет экономить память, так как одна и та же строка не создается заново, а переиспользуется.

Создание строки литералом (`String s = "Hello"`) помещает ее в пул строк (если такой строки там нет).

Создание строки через `new` (`String s = new String("Hello")`) всегда создает новый объект вне пула.

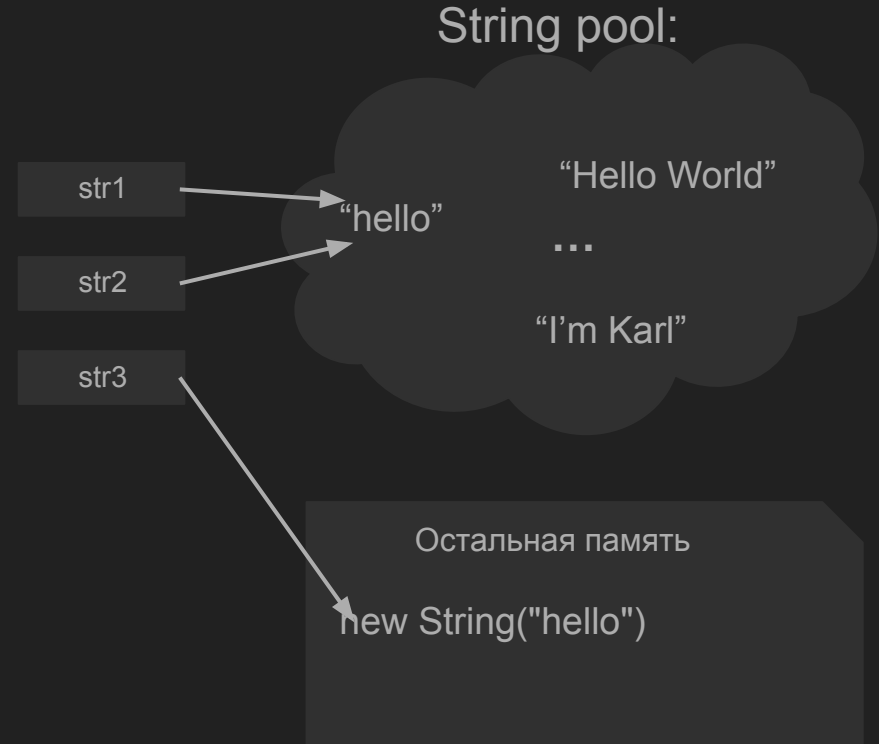


Сравнение строк

```
package l24;

public class StringComparing {
    public static void main(String[] args) {
        String str1 = "hello";
        String str2 = "hello";
        String str3 = new String("hello");

        // true потому что переменные
        // указывают на один и тот же объект
        System.out.println(str1 == str2);
        // false потому что переменные
        // указывают на разные объекты
        System.out.println(str1 == str3);
        // true потому что сравниваем
        // значения строк
        System.out.println(str1.equals(str3));
    }
}
```





StringBuilder

```
package l24;

public class StringBuilderExample {
    public static void main(String[] args) {
        // Создаем объект StringBuilder
        StringBuilder shoppingList;
        shoppingList = new StringBuilder();

        // Добавляем пункты в список покупок
        shoppingList.append( "Молоко" );
        shoppingList.append( ", Яйца" );
        shoppingList.append( ", Хлеб" );
        shoppingList.append( ", Шоколад" );

        // Выведем итоговый список покупок
        System.out.println(
            "Список покупок: " +
            shoppingList.toString()
        );
    }
}
```

StringBuilder - это класс, предназначенный для работы со строками, которые нужно часто изменять.

В отличие от класса String, объекты StringBuilder изменяемы.

Это делает StringBuilder идеальным выбором для операций, требующих множественной модификации строк, так как это снижает нагрузку на вашу программу.



Метод toString

```
package l24;

class Soda {
    private String name; // Название напитка
    private int volume; // Объем напитка в миллилитрах

    // тут должен быть конструктор

    // Переопределение метода toString для представления
    // объекта Soda в удобочитаемом формате
    @Override
    public String toString() {
        return "Soda{" + "name='" + name + "'" +
            ", volume=" + volume + "ml" + '}';
    }

    public static void main(String[] args) {
        // Создаем объект cola класса Soda
        Soda cola = new Soda("Cola", 500);
        // Печатаем информацию о cola, автоматически
        // вызывается метод toString
        System.out.println(col.toString());
        System.out.println(col); // то же самое
    }
}
```

Метод toString в Java предназначен для возвращения строкового представления объекта.

В классе Object этот метод возвращает строку, которая состоит из имени класса объекта, символа @ и его хеш-кода в шестнадцатеричном представлении.

Хотя это предоставляет базовую информацию об объекте, оно обычно бесполезно.

Переопределяя toString, вы можете предоставить более понятное и подробное описание состояния объекта, что облегчает отладку и логирование, а также улучшает взаимодействие с пользователем программы.



Методы equals и hashCode

```
package l24;

class Person {
    private String name;
    private int age;
    // тут должен быть конструктор
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
            return false;

        Person person = (Person) obj;
        return age == person.age && (
            name == null ? person.name == null :
                name.equals(person.name)
        );
    }

    @Override
    public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + age;
        return result;
    }
}
```

Метод equals в Java используется для проверки равенства между двумя объектами.

По умолчанию, реализация equals в классе Object сравнивает ссылки на объекты, то есть проверяет, указывают ли две ссылки на один и тот же объект в памяти.

Однако, часто требуется сравнивать объекты по их содержимому, а не по их ссылкам. В таких случаях метод equals необходимо переопределить.



Методы equals и hashCode

```
package l24;

class Person {
    private String name;
    private int age;
    // тут должен быть конструктор
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
            return false;

        Person person = (Person) obj;
        return age == person.age && (
            name == null ? person.name == null :
                name.equals(person.name)
        );
    }

    @Override
    public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + age;
        return result;
    }
}
```

При переопределении equals, важно следовать контракту equals, который требует:

- Рефлексивность: объект должен быть равен самому себе (x.equals(x) всегда true).
- Симметричность: если x.equals(y) true, то и y.equals(x) должен быть true.
- Транзитивность: если x.equals(y) и y.equals(z) оба true, то и x.equals(z) должен быть true.
- Согласованность: если информация, используемая в сравнении объектов, не изменяется, то многократные вызовы x.equals(y) должны возвращать одинаковый результат.
- Ненулевая ссылка: для любого ненулевого значения x, x.equals(null) должен быть false.



Методы equals и hashCode

```
package 124;

class Person {
    private String name;
    private int age;
    // тут должен быть конструктор
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
            return false;

        Person person = (Person) obj;
        return age == person.age && (
            name == null ? person.name == null :
                name.equals(person.name)
        );
    }

    @Override
    public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + age;
        return result;
    }
}
```

Когда вы переопределяете equals, важно также переопределить hashCode.

Это требуется для соблюдения общего контракта метода hashCode, который гласит, что если два объекта равны согласно методу equals, то вызов метода hashCode для этих объектов должен возвращать одинаковое целочисленное значение.

Это правило необходимо для корректной работы объектов в коллекциях, основанных на хеш-таблицах, таких как HashMap и HashSet.



Домашнее задание

доделать домашку 23

Добавить во все свои классы-родители переопределение метода equals и hashCode и написать модульные тесты (junit) для проверки по контрактам соответствующих методов (два последних слайда в презентации).

Вариант 1: на основе 23 домашки основанной на 17 домашке

Вариант 2: на основе 23 домашки основанной на Покемонах



The end