



# План занятия

1. Что такое объекты?
2. Зачем нужны классы?
3. Создаём хомяка
4. Практикуемся с объектами
5. Конструкторы
6. Рефакторинг финансового приложения



Что такое объекты?



# Что такое объекты?

Вы уже познакомились с переменными и массивами для хранения значений. Умеете использовать циклы, условные выражения и методы. Эта тема посвящена ещё одной важной составляющей Java-программирования — **объектам**.

Существуют разные подходы к написанию программ. Набор идей и понятий, которые помогают упорядочивать и писать код, называется **парадигмой программирования**. Программы, написанные в одной парадигме, по своей структуре похожи друг на друга. Даже в тех случаях, если они написаны на разных языках.



# Что такое объекты?

Одна из самых распространённых парадигм в современной разработке — **объектно-ориентированное программирование (ООП)**. Код на Java пишут именно в таком стиле, а сам язык принято называть объектно-ориентированным.

Основная задача ООП — сделать сложный код проще. Для этого программу разбивают на независимые блоки — объекты. Внутри каждого объекта хранится набор переменных и методов, сгруппированных вместе. Представьте, что часть кода положили в коробку и закрыли крышкой. Такая коробка и есть объект. Однако, в отличие от коробки, объекты не пылятся где-то на шкафу, а активно взаимодействуют внутри программы.



# Что такое объекты?

С помощью объектов можно описать как материальные сущности — кошку, машину, человека, так и абстрактные понятия — счёт в банке, категорию трат в финансовом приложении или заказ в интернет-магазине.

Разберем понятие объекта на примере хомяка Байта. Во-первых, у любого объекта есть **свойства**, которые его характеризуют. Их называют **атрибутами** или **полями**. У Байта есть такие свойства: имя, вес, возраст и цвет. Во-вторых, у объекта есть набор действий (функций), которые он выполняет — это его **методы**. Байт, к примеру, ест, спит, бегают в колесе и прячет еду за щёки.



Программы в ООП состоят из большого числа объектов, у каждого из которых своя функциональность (совокупность методов). За счет методов объекты взаимодействуют между собой. К примеру, в одной программе может быть не только хомяк, но и его хозяйка — студентка Настя. Каждый из объектов должен уметь реагировать на действия другого. Если хомяк сидит с грустным видом возле пустой миски, то стоит его покормить. Поэтому у Насти нужно создать метод «Покормить питомца».





# Что такое объекты?

**Метод** - действия, которые объект может выполнять.

**Поле и атрибут** - свойства, описывающие конкретный объект.

**Объект** - набор полей и методов, сгруппированных вместе, для представления какой-то сущности.





# Что такое объекты?

Таким образом, чтобы написать программу в объектно-ориентированном стиле, определите, какие понадобятся объекты и как они должны взаимодействовать. Затем тщательно проработайте каждый объект: опишите его свойства и методы. Из получившихся блоков, как из кирпичиков, вырастет программа.

Разделение кода на логические блоки сильно упрощает процесс его написания. Так что без умения создавать объекты в коде Java-программисту не обойтись.



# Зачем нужны классы?

# Зачем нужны классы?

Вы уже знаете, что такое объекты. Чтобы научиться их создавать, освоим еще одно понятие ООП — **классы**. Весь код на Java пишется внутри классов: чтобы запустить программу, нужно описать класс со специальным методом **main**.

```
public class Practice {  
    public static void main(String[] args) {  
        // Ваш код  
    }  
}
```




# Зачем нужны классы?

Однако основное предназначение класса — создание структуры объекта. Ранее мы говорили о том, что, прежде чем создавать объект, нужно подробно описать его свойства и методы. Так вот, если вы это сделаете, то как раз получите класс — шаблон для создания объекта. Если представить, что **объект** — свеча, то **класс** — форма для ее отливки. За счет одной формы можно отлить неограниченное количество свечей. Так же на основе одного класса можно создать неограниченное число объектов.

# Зачем нужны классы?


Создание класса начинается с его объявления. Для этого используется слово **class**, после которого с заглавной буквы идет его имя: у нас будет класс **Hamster**.

```
// Объявляем класс с именем Hamster (англ. «хомяк»)  
public class Hamster {  
  
}
```



Когда класс объявлен, можно приступить к его описанию. Внутри фигурных скобок нужно объявить поля (свойства объекта) и методы (функции) — это **тело класса**. Начинать принято с полей. В классе **Hamster** мы укажем — имя, возраст, вес и цвет. Поля — это переменные, чтобы их объявить, используются хорошо знакомые вам типы данных **String** и **int**:

```
public class Hamster {  
    // Поля  
    String name = "Байт";  
    int age = 2;  
    int weight = 350;  
    String color = "Рыжий";  
}
```




У всех объектов-хомяков класса **Hamster** будет имя Байт. Им будет по два года. Они будут весить 350 грамм и все будут рыжими.

Следующий шаг — описать методы будущих объектов. Нужно зафиксировать в классе, что его объекты-хомяки должны уметь есть, бегать в колесе и прятать семечки за щеки. У всех методов класса **Hamster** будет тип **void**. Также при их объявлении мы не будем использовать служебные слова **public** и **static**. Значения полей объекта можно будет передать методам в качестве аргументов.

```
public class Hamster {  
    // Поля  
    String name = "Байт"; // Имя  
    int age = 2; // Возраст  
    String color = "Рыжий"; // Цвет  
    int weight = 350; // Вес в граммах  
  
    // Есть  
    void eat(int foodWeight) {  
        weight = weight + foodWeight; // Если покормить Байта, то он немного прибавит в весе  
    }  
  
    // Бегать в колесе  
    void runInWheel() {  
        System.out.println("Бегу-бегу-бегу!");  
        weight = weight - 5; // Байт может сбросить вес, бегая в колесе  
    }  
  
    // Прятать семечки  
    void hideSeeds(int seedWeight) {  
        weight = weight + seedWeight; // Семечки за щекой сделают Байта неповоротливым  
        System.out.println("Зимой не заголодаю.");  
    }  
}
```





В классе `Hamster` появилось три метода `eat(int foodWeight)`, `runInWheel()` и `hideSeeds(int seedWeight)`. Все объекты-хомяки этого класса будут прибавлять вес после еды и семечек и худеть на 5 грамм после пробежки.



# Вопрос

Верное определение класса.

- A. Класс — это объект.
- B. Класс — это заготовка для объекта.
- C. Класс — это то, где пишем `main()` и всё работает.
- D. Класс содержит поля



# Ответ

В. Класс — это заготовка для объекта.  
Да, всё так, на его основе можно создать объект.



## Задача

Пришло время добавить класс в финансовое приложение. Сейчас весь код описан в одном классе **Practice**: и конвертация валют, и работа с тратами, и взаимодействие с пользователем. В таком коде становится сложно ориентироваться. Вы уже разбили его на методы, с помощью классов можно сделать его ещё проще. Создайте пустой класс **Converter**. Позже в нём будет храниться вся логика, связанная с конвертацией валют.

## Задача 2

Теперь добавьте классу **Converter** поля и присвойте им значения: курс доллара, евро и иены. Имена полей будут совпадать с теми, что вы использовали ранее: **rateUSD**, **rateEUR**, **rateJPY**.

```
public class Converter {  
    //доллар = 450;  
    //евро = 500;  
    //йена = 3;  
}
```

# Решение

```
public class Converter {  
    double rateUSD = 450;  
    double rateEUR = 500;  
    double rateJPY = 3;  
}
```

## Задача 2

Вам нужно, чтобы конвертер умел рассчитать соотношение валют. Для этого объявите метод: **convert**. Этот метод должен принимать такие параметры, как сумма в тенге (**double tenges**) и код валюты (**int currency**).

```
public class Converter {  
    double rateUSD = 450;  
    double rateEUR = 500;  
    double rateJPY = 3;  
  
    // Здесь объявляете метод {  
    //     Тело метода  
    // }  
}
```

# Решение

```
public class Converter {
    double rateUSD = 450;
    double rateEUR = 500;
    double rateJPY = 3;

    void convert(double tenges, int currency) {
        if (currency == 1) {
            System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);
        } else if (currency == 2) {
            System.out.println("Ваши сбережения в евро: " + tenges / rateEUR);
        } else if (currency == 3) {
            System.out.println("Ваши сбережения в йенах: " + tenges / rateJPY);
        } else {
            System.out.println("Неизвестная валюта");
        }
    }
}
```





# Создаём хомяка



## Создаём хомяка


На основе класса создадим объект — конкретного хомяка. Класс **Hamster** полностью готов.

```
public class Hamster {  
    // Поля  
    String name = "Байт"; // Имя  
    int age = 2; // Возраст  
    String color = "Рыжий"; // Цвет  
    int weight = 350; // Вес в граммах  
  
    // Есть  
    void eat(int foodWeight) {  
        weight = weight + foodWeight; // Если покормить Байта, то он немного прибавит в весе  
    }  
  
    // Бегать в колесе  
    void runInWheel() {  
        System.out.println("Бегу-бегу-бегу!");  
        weight = weight - 5; // Байт может сбросить вес, бегая в колесе  
    }  
  
    // Прятать семечки  
    void hideSeeds(int seedWeight) {  
        weight = weight + seedWeight; // Семечки за щекой сделают Байта неповоротливым  
        System.out.println("Зимой не заголодаю.");  
    }  
}
```

## Создаём хомяка

Наш объект будет, как и другие значения в программе, храниться в переменной. Нужно её объявить. Чтобы это сделать, требуется указать тип переменной. Раньше мы использовали такие типы данных, как `int`, `double` и `String`. Тут действует тот же принцип, только вместо `int` или `String` будет `Hamster`. Каждый новый класс создаёт новый тип данных. Поэтому имя класса становится типом переменной. Вот как это выглядит:

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster bite; // Объявили переменную с типом Hamster  
    }  
}
```




Для новой переменной **Hamster** мы придумали имя **bite**. По этому имени можно будет обращаться к объекту. Однако объект всё ещё не создан, потому что переменной не присвоено значение. Представьте, будто вы решили завести хомяка, купили для него клетку и повесили на неё табличку «Байт». Ваш следующий шаг теперь — посадить в эту клетку с табличкой реального грызуна.

Для того чтобы присвоить переменной **bite** значение нового объекта, понадобится слово **new** (с ним вы уже сталкивались при изучении массивов), имя класса и пара круглых скобок, как при вызове метода, — **new Hamster()**. Без **new** не обойтись — объекты создаются только с его помощью.

```
Hamster bite = new Hamster(); // Присвоили переменной значение нового объекта-хомяка
```

Объект-хомяк создан! Он хранится в переменной **bite** и обладает всеми атрибутами и методами своего класса. У него есть имя — Байт, его вес — 350 грамм, возраст — 2 года, а цвет — «Рыжий». Он умеет есть, спать, бегать в колесе и прятать еду за щёки.





Взаимодействие с новым объектом происходит через его методы. Например, чтобы хомяк побегал в колесе, нужно вызвать метод `runInWheel()`. Это происходит с помощью **точечной нотации** — используется точка после имени объекта:


```
public class Practice {  
    public static void main(String[] args) {  
        Hamster bite = new Hamster();  
        bite.runInWheel(); // Вызов метода  
    }  
}
```



# Результат

Бегу-бегу-бегу!





Метод `runInWheel()`, как вы помните, не только печатает текст, но и уменьшает вес хомяка на 5 грамм. В этом можно убедиться, если вывести значения поля `weight` до и после того, как хомяк побегаёт в колесе. Обращение к полям объекта также происходит с помощью **точечной нотации** — получится `bite.weight`.

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster bite = new Hamster();  
        System.out.println("Вес хомяка до пробежки: " + bite.weight);  
        bite.runInWheel();  
        System.out.println("Вес хомяка после пробежки: " + bite.weight);  
    }  
}
```



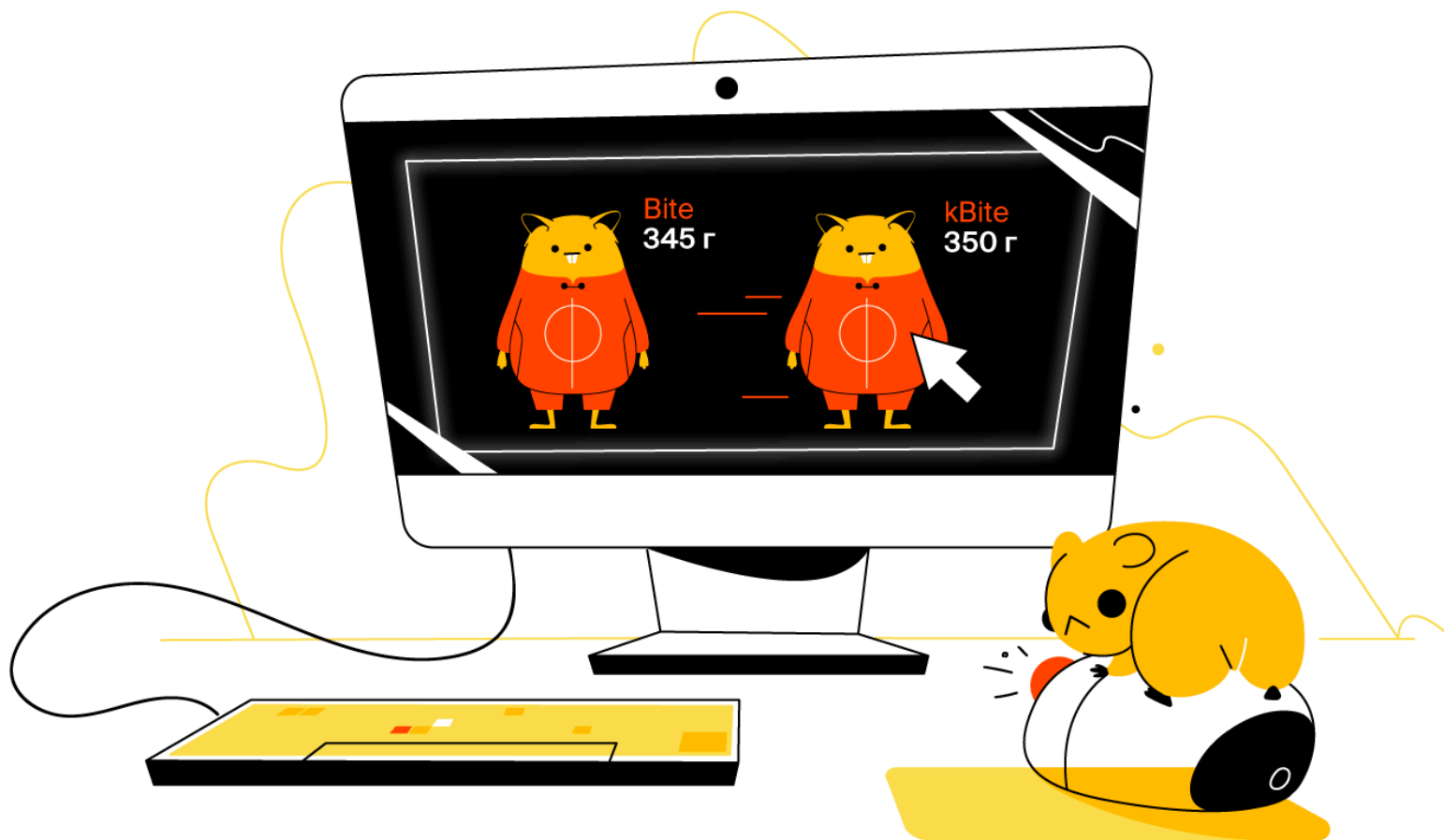
# Результат


Вес хомяка до пробежки: 350

Бегу-бегу-бегу!

Вес хомяка после пробежки: 345

Поздравляем, вы создали объект — хомяка, заставили его побегать и немного похудеть. Важно запомнить: когда мы меняем свойства объекта, то на классе эти изменения не отразятся. Это означает, что следующий хомяк, новый объект (например, `kBite`), будет снова весить 350 грамм.





Теперь представим, что вам захотелось дать хомяку новое имя. Чтобы поменять базовое значение поля объекта, указанное в классе, обратимся к нему и присвоим этому полю новое значение (например, Хомка). Выведем изменения на экран:

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster bite = new Hamster();  
  
        System.out.println("Раньше хомяка звали: " + bite.name); // Выведем на экран изначальное имя  
        bite.name = "Хомка"; // Присвоим полю name новое значение  
        System.out.println("Теперь хомяка зовут: " + bite.name); // А теперь посмотрим, что получилось  
    }  
}
```



# Результат

Раньше хомяка звали: Байт  
Теперь хомяка зовут: Хомка

Теперь вы знаете, как создать объект на основе класса, умеете менять его базовые атрибуты и взаимодействовать с ним.

# Задача

Программа не работает — мы пытаемся обратиться к объекту, но он ещё не создан. Исправьте это.

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster homka = ...  
  
        System.out.println("Имя: " + homka.name);  
    }  
}
```

# Решение

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster homka = new Hamster();  
  
        System.out.println("Имя: " + homka.name);  
    }  
}
```

## Задача 2

Хотим знать о хомяке больше: добавьте информацию о нём — возраст (англ. *age*), вес (англ. *weight*) и цвет (англ. *color*).

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster homka = new Hamster();  
  
        System.out.println("Имя: " + homka.name);  
        System.out.println("Возраст: " + ...);  
        System.out.println("Вес: " + ...);  
        System.out.println("Цвет: " + ...);  
    }  
}
```



# Решение

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster homka = new Hamster();  
  
        System.out.println("Имя: " + homka.name);  
        System.out.println("Возраст: " + homka.age);  
        System.out.println("Вес: " + homka.weight);  
        System.out.println("Цвет: " + homka.color);  
    }  
}
```

## Задача 3

Переменная называется **homka**, но хомяка все еще зовут Байт. Измените имя хомяка на Хомку, а заодно отредактируйте и другие атрибуты: пусть ему будет 1 год, он будет весить 420 грамм, а его цвет будет не рыжий, а чёрный.

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster homka = new Hamster();  
        // Присвойте полям новые значения  
        ...  
  
        System.out.println("Имя: " + homka.name);  
        System.out.println("Возраст: " + homka.age);  
        System.out.println("Вес: " + homka.weight);  
        System.out.println("Цвет: " + homka.color);  
    }  
}
```

# Решение

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster homka = new Hamster();  
        homka.name = "Хомка";  
        homka.age = 1;  
        homka.weight = 420;  
        homka.color = "Чёрный";  
  
        System.out.println("Имя: " + homka.name);  
        System.out.println("Возраст: " + homka.age);  
        System.out.println("Вес: " + homka.weight);  
        System.out.println("Цвет: " + homka.color);  
    }  
}
```

## Задача 4

Хомка голодный! Покормите его с помощью метода **eat**. В качестве аргумента передайте число 15. После этого проверьте значение его веса снова, чтобы убедиться, что он поужинал.

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster homka = new Hamster();  
        homka.name = "Хомка";  
        homka.age = 1;  
        homka.weight = 420;  
        homka.color = "Чёрный";  
  
        System.out.println("Имя: " + homka.name);  
        System.out.println("Возраст: " + homka.age);  
        System.out.println("Вес: " + homka.weight);  
        System.out.println("Цвет: " + homka.color);  
        ...  
        System.out.println("Новый вес: " + ...);  
    }  
}
```

# Решение

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster homka = new Hamster();  
        homka.name = "Хомка";  
        homka.age = 1;  
        homka.weight = 420;  
        homka.color = "Чёрный";  
  
        System.out.println("Имя: " + homka.name);  
        System.out.println("Возраст: " + homka.age);  
        System.out.println("Вес: " + homka.weight);  
        System.out.println("Цвет: " + homka.color);  
        homka.eat(15);  
        System.out.println("Новый вес: " + homka.weight);  
    }  
}
```

## Задача 5

В прошлый раз вы описали класс конвертера валют **Converter**. Дополните код финансового приложения так, чтобы при запуске программы работала конвертация валют. Для этого потребуется создать конвертер-объект и вызвать его метод.

[https://github.com/practicetasks/java\\_tasks/tree/main/classes\\_and\\_objects/task\\_1](https://github.com/practicetasks/java_tasks/tree/main/classes_and_objects/task_1)



# Решение

<https://gist.github.com/practicetasks/975cd9a2c96c28f429628253baa23689>



# Практикуемся с объектами





## Практикуемся с объектами

Еще немного потренируемся работать с объектами. Как вы знаете, на основе одного класса можно создать неограниченное число объектов. Представим, что вы решили завести не одного, а нескольких хомяков.

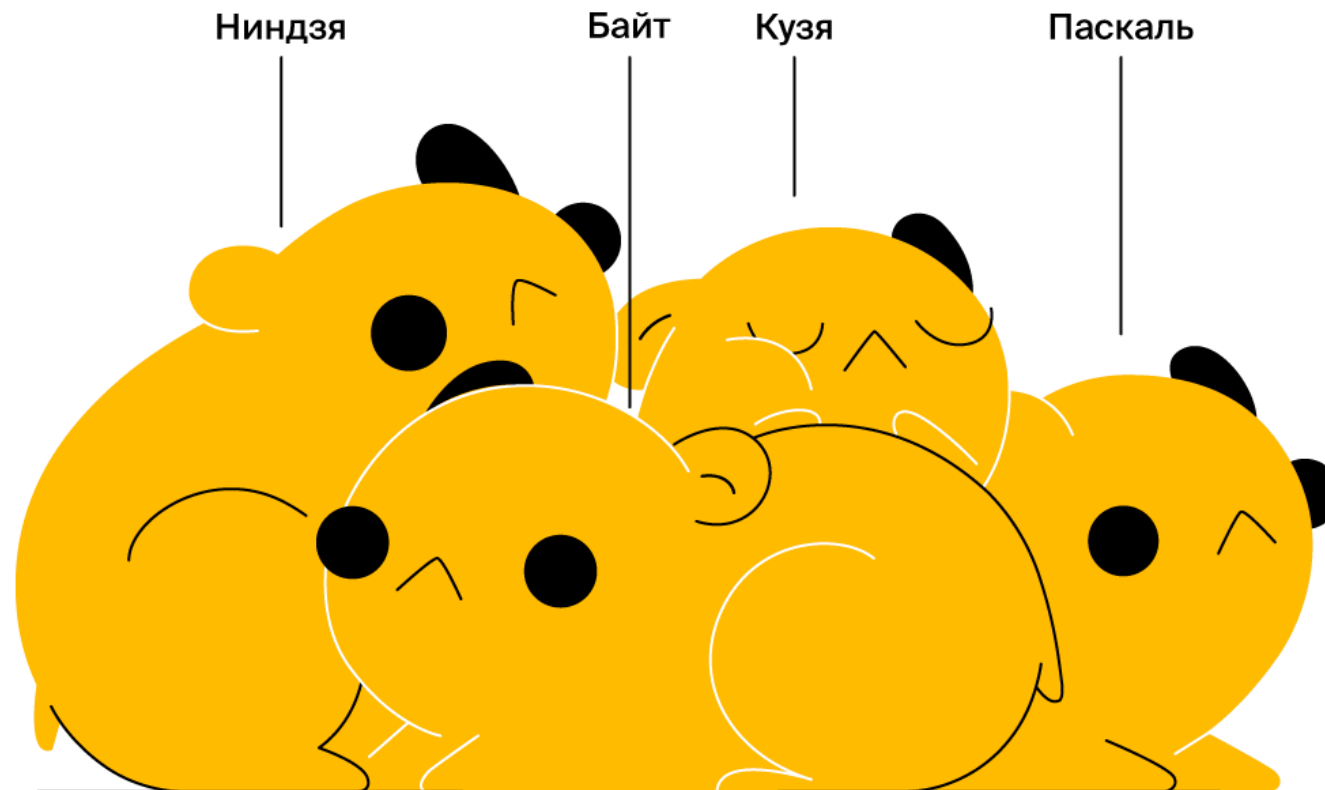
```
public class Hamster {  
    //Поля  
    String name = "Байт"; // Имя  
    int age = 2; // Возраст  
    String color = "Рыжий"; // Цвет  
    int weight = 350;  
  
    // Методы  
    void eat(int foodWeight) {  
        weight = weight + foodWeight;  
    }  
  
    // Бегать в колесе  
    void runInWheel() {  
        System.out.println("Бегу-бегу-бегу!");  
        weight = weight - 5;  
    }  
  
    // Прятать семечки  
    void hideSeeds(int seedWeight) {  
        System.out.println("Зимой не заголодаю.");  
        weight = weight + seedWeight;  
    }  
}
```



Создадим четыре новых объекта:

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster bite = new Hamster();  
        Hamster ninja = new Hamster();  
        Hamster kuzya = new Hamster();  
        Hamster paskal = new Hamster();  
    }  
}
```

У вас теперь четыре питомца. Все они идентичны друг другу: у них полностью совпадают имя, вес, возраст и цвет. Однако это разные объекты. Или **экземпляры класса** (англ. instance) — этот термин также часто используется в ООП для обозначения объектов.



```
public class Practice {  
    public static void main(String[] args) {  
        Hamster bite = new Hamster(); // Имя Байта менять не нужно  
        Hamster ninja = new Hamster();  
        ninja.name = "Ниндзя";  
  
        Hamster kuzya = new Hamster();  
        kuzya.name = "Кузя";  
  
        Hamster paskal = new Hamster();  
        paskal.name = "Паскаль";  
  
        System.out.println("Первого хомяка зовут " + bite.name);  
        System.out.println("Второго хомяка зовут " + ninja.name);  
        System.out.println("Третьего хомяка зовут " + kuzya.name);  
        System.out.println("Четвертого хомяка зовут " + paskal.name);  
    }  
}
```



# Результат

Первого хомяка зовут Байт  
Второго хомяка зовут Ниндзя  
Третьего хомяка зовут Кузя  
Четвертого хомяка зовут Паскаль

Сделаем грызунов еще более непохожими друг на друга — пусть у каждого будут свои значения возраста, веса и цвета:

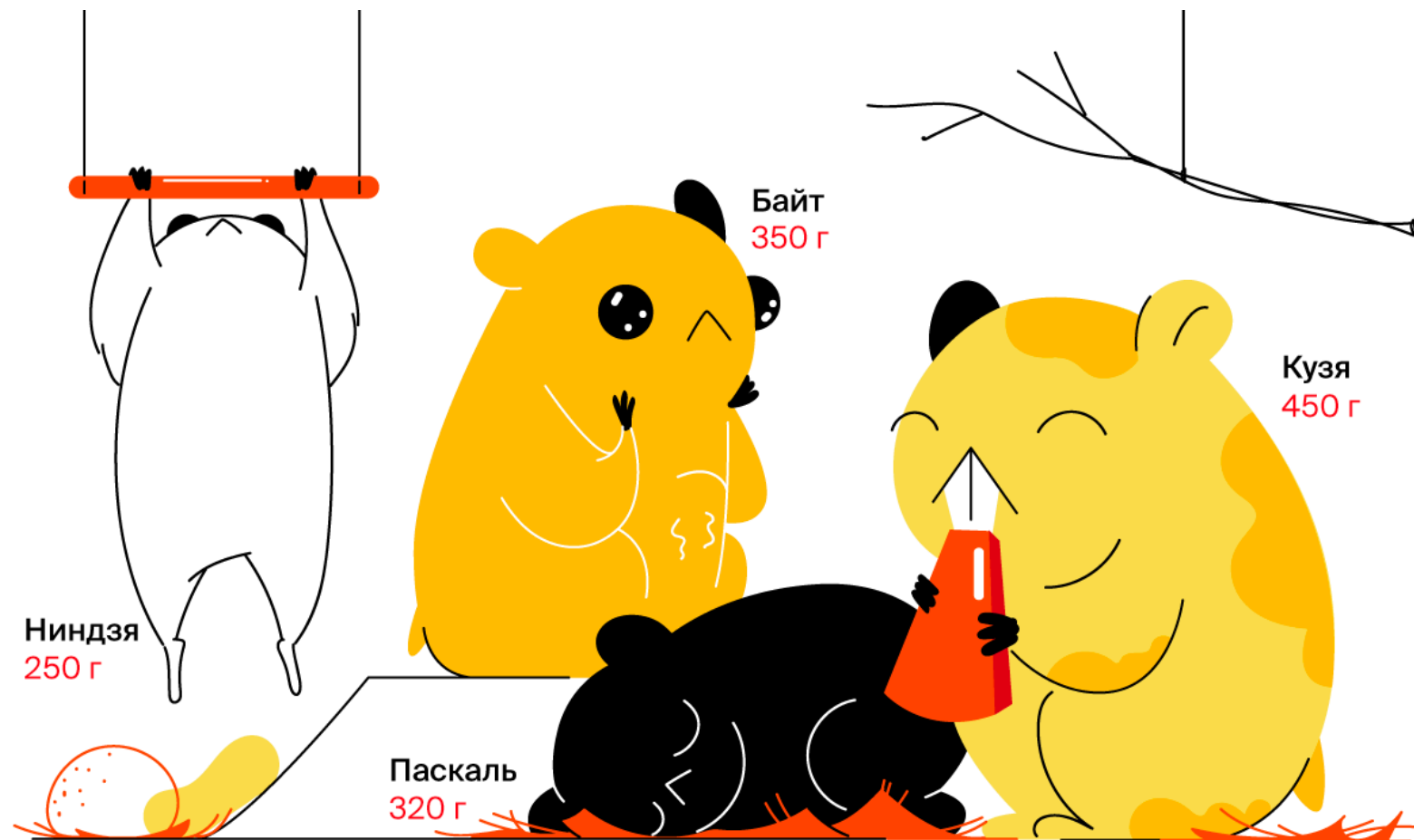
```
Hamster bite = new Hamster(); // Свойства Байта менять не нужно
```

```
Hamster ninja = new Hamster();  
ninja.name = "Ниндзя";  
ninja.age = 1;  
ninja.weight = 250;  
ninja.color = "Белый";
```

```
Hamster kuzya = new Hamster();  
kuzya.name = "Кузя";  
kuzya.age = 3;  
kuzya.weight = 450;  
kuzya.color = "Пятнистый";
```

```
Hamster paskal = new Hamster();  
paskal.name = "Паскаль";  
paskal.age = 4;  
paskal.weight = 320;  
paskal.color = "Чёрный";
```

Получилось четыре разных хомяка: рыжий двухлетний Байт весом 350 грамм, белый годовалый Ниндзя весом 250 грамм, пятнистый трехлетний Кузя, вес которого 450 грамм, и черный четырехлетний Паскаль весом 320 грамм.







# Вопрос

Какие из этих утверждения верные:

- A. У разных объектов, созданных на основе одного класса, одинаковая структура — набор полей и методов.
- B. У разных объектов, созданных на основе одного класса, всегда совпадают значения полей.
- C. Значение полей одного объекта зависит от значения полей другого объекта.
- D. Значение полей одного объекта не зависит от значения полей другого объекта.



## Ответ

А. У разных объектов, созданных на основе одного класса, одинаковая структура — набор полей и методов.

Все объекты одного класса имеют одинаковый набор полей и методов.

В. У разных объектов, созданных на основе одного класса, всегда совпадают значения полей.

Значения полей экземпляров одного класса можно менять, а значит они могут как совпадать, так и отличаться.

С. Значение полей одного объекта зависит от значения полей другого объекта.  
Объекты не зависят друг от друга.

Д. Значение полей одного объекта не зависит от значения полей другого объекта.  
Если вы меняете свойства одного объекта, то это не отражается на остальных.



# Конструкторы



# Конструкторы

В ходе прошлого урока вы научились создавать несколько объектов одного класса и менять у них значения полей. Код получился таким:



```
Hamster bite = new Hamster(); // Свойства Байта менять не нужно
```

```
Hamster ninja = new Hamster();  
ninja.name = "Ниндзя";  
ninja.age = 1;  
ninja.weight = 250;  
ninja.color = "Белый";
```

```
Hamster kuzya = new Hamster();  
kuzya.name = "Кузя";  
kuzya.age = 3;  
kuzya.weight = 450;  
kuzya.color = "Пятнистый";
```

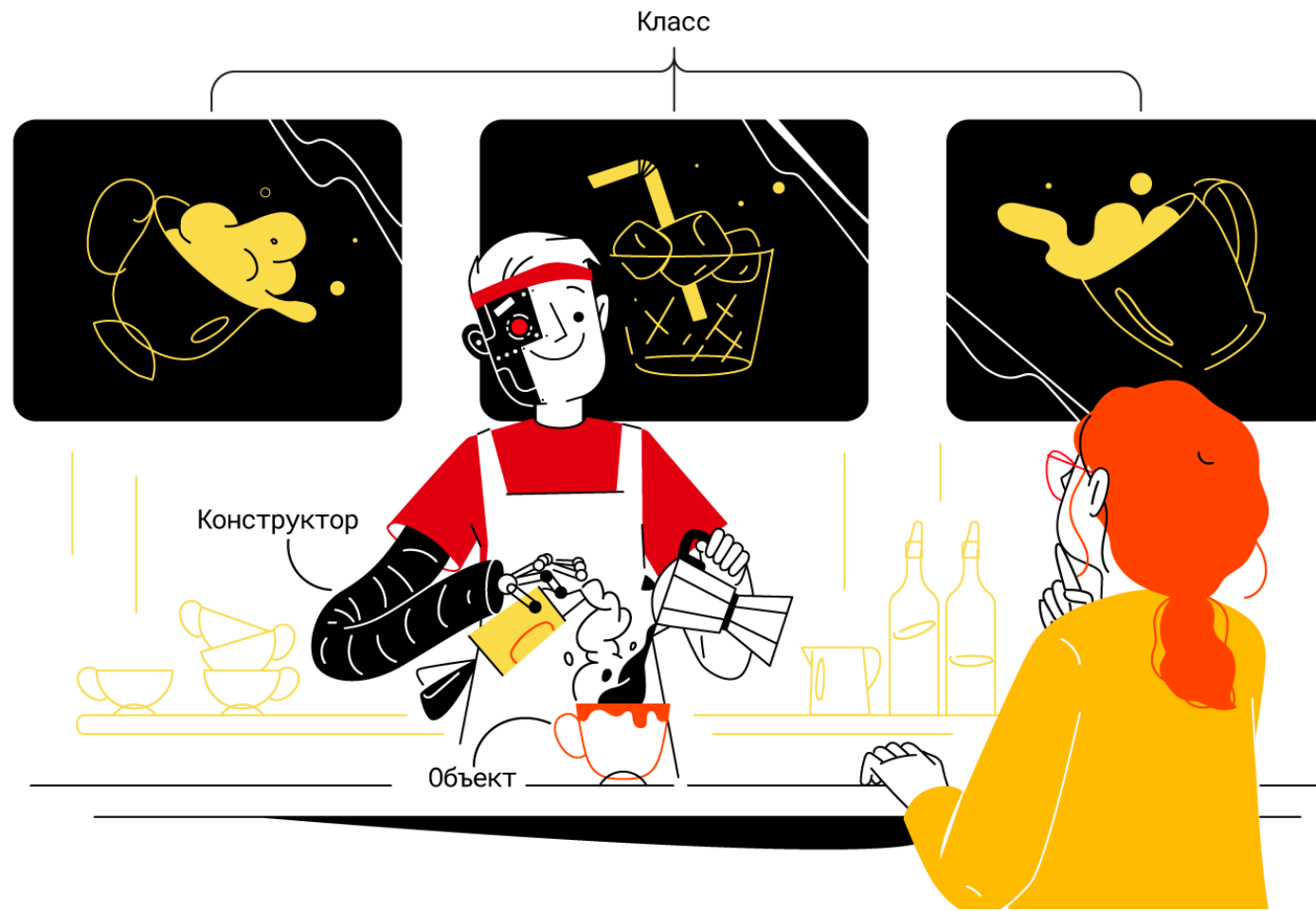
```
Hamster paskal = new Hamster();  
paskal.name = "Паскаль";  
paskal.age = 4;  
paskal.weight = 320;  
paskal.color = "Чёрный";
```




# Конструкторы

Теперь представьте, что вам нужно не четыре объекта, а пару десятков, и свойств у каждого из них будет больше, чем у хомяков. Придётся сначала создать все объекты, а после присвоить полям каждого новые значения. Это долго и скучно, а программа получится громоздкой. Хочется сразу получить объекты с нужными значениями полей. Для решения этой задачи в Java есть специальный инструмент — **конструктор**.

Конструктор позволяет создавать новые объекты класса. Если класс — это чертёж, то конструктор — это станок, который штампует экземпляры по заданным параметрам. Это как бариста в кофейне: по вашей просьбе он может сделать капучино по классическому рецепту, а может добавить сироп, корицу и кокосовую стружку.





На самом деле вы уже пользовались конструктором при создании новых объектов. Когда вы указывали имя класса с парой круглых скобок после слова `new`, то вызывали **конструктор по умолчанию**. Если в классе нет других конструкторов, Java автоматически добавит такой конструктор в каждый класс. Хотя вы и не знали о нём, именно он создавал новый экземпляр класса:


```
public class Practice {  
    public static void main(String[] args) {  
        Hamster bite = new Hamster(); // Здесь вызывается конструктор по умолчанию  
    }  
}
```






# Конструкторы

**Конструктор по умолчанию** позволяет создавать объекты с одинаковыми значениями полей. Когда в прошлых уроках вы описывали в классе структуру объектов, то присваивали полям готовое значение. В результате вы получали одинаковые экземпляры — все хомяки были рыжими двухлетними Байтами весом по 350 грамм. Это происходило благодаря конструктору по умолчанию.




Поскольку мы хотим, чтобы все хомяки были разными, нам нужен **конструктор с параметрами** — ему можно «заказать» объекты с конкретными свойствами. Конструктор с параметрами объявляется внутри класса. Сначала создадим заготовку — пустой конструктор. Это несложно. После полей класса указываем имя конструктора — оно совпадает с именем класса — и пару круглых скобок:

```
public class Hamster {  
    // Поля  
    String name = "Байт"; // Имя  
    int age = 2; // Возраст  
    String color = "Рыжий"; // Цвет  
    int weight = 350;  
  
    // Пустой конструктор. Java создает такой автоматически для типовых объектов  
    Hamster() {  
    }  
}
```



Когда используется конструктор с параметрами, то значения полям присваиваются внутри него. Поэтому на уровне класса нужно оставить только объявление полей. В коде это выглядит так:

```
public class Hamster {  
    // Объявили поля, но не присвоили им значения  
    String name;    // Имя  
    int age;        // Возраст  
    int weight;     // Вес в граммах  
    String color;   // Цвет  
  
    // Конструктор принимает 4 параметра  
    Hamster(String hamsterName, int hamsterAge, int hamsterWeight, String hamsterColor) {  
        name = hamsterName;  
        age = hamsterAge;  
        weight = hamsterWeight;  
        color = hamsterColor;  
    }  
}
```




Значения полям присваиваются внутри конструктора. Теперь при создании нового объекта, после слов **new Hamster**, будет запускаться конструктор с параметрами, внутри круглых скобок которого вы можете сразу указать нужные атрибуты объекта. Создавать разных хомяков стало проще:

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster volt = new Hamster("Вольт", 2, 340, "Рыжий");  
        Hamster bosya = new Hamster("Бося", 1, 300, "Чёрный");  
    }  
}
```



# Конструкторы

В результате созданы два хомяка: рыжий двухлетний хомяк весом 340 грамм по имени Вольт и Бося — годовалый чёрный грызун весом 300 грамм. Для их создания понадобилось всего две строки.



Важная деталь: когда вы объявили в классе конструктор с параметрами, программа перестает добавлять конструктор по умолчанию. Если попробовать создать новый объект и не передать параметры — произойдёт ошибка. Такой код не запустится:

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster bite = new Hamster(); // В этой строке произойдёт ошибка  
    }  
}
```

# Результат

Practice:3:21

```
java: constructor Hamster in class Hamster cannot be applied to given types;  
  required: java.lang.String,int,int,java.lang.String  
  found:    no arguments  
  reason: actual and formal argument lists differ in length
```

Программа сообщает, что вы пытаетесь обратиться к конструктору по умолчанию, однако в классе теперь есть только конструктор с параметрами. Чтобы исправить ошибку, требуется вызвать его.



# Конструкторы

В одном классе может быть несколько конструкторов. Имена у них будут совпадать. Главное, чтобы отличались их сигнатуры — количество и типы параметров. Тогда программа сможет понять, какой именно конструктор вызвать.

Представим, что вы хотите создавать хомяков одного цвета. Остальные параметры: имя, возраст и вес — у них должны отличаться, а окрас у всех должен быть рыжий. Вам потребуется объявить новый конструктор, который будет принимать значения только имени, возраста и веса хомяка, а за переменной цвета будет закреплено постоянное значение.



Это выглядит так:

```
public class Hamster {
    String name;    // Имя
    int age;        // Возраст
    int weight;    // Вес в граммах
    String color;   // Цвет

    // Конструктор № 1 создаёт только рыжих хомяков
    Hamster(String hamsterName, int hamsterAge, int hamsterWeight) {
        name = hamsterName;
        age = hamsterAge;
        weight = hamsterWeight;
        color = "Рыжий"; // Закрепили за переменной постоянное значение
    }

    // Конструктор № 2 создаёт разных хомяков
    Hamster(String hamsterName, int hamsterAge, int hamsterWeight, String hamsterColor) {
        name = hamsterName;
        age = hamsterAge;
        weight = hamsterWeight;
        color = hamsterColor;
    }
}
```

По количеству параметров при вызове конструктора, Java понимает, какой из них нужен. Если при создании объекта используется три значения — имя, возраст и, вес, то вы получите рыжего хомяка (вызван Конструктор № 1). Чтобы убедиться в этом, выведем цвет хомяка на экран:

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster ginger = new Hamster("Рыжик", 3, 250);  
        System.out.println("Цвет хомяка: " + ginger.color);  
    }  
}
```



# Результат

Цвет хомяка: Рыжий

Получить такой же результат можно и по-другому: если инициализировать поле **color** при его объявлении.

```
public class Hamster {
    String name;    // Имя
    int age;        // Возраст
    int weight;     // Вес в граммах
    String color = "Рыжий"; // Цвет

    // Конструктор создаёт только рыжих хомяков
    Hamster(String hamsterName, int hamsterAge, int hamsterWeight) {
        name = hamsterName;
        age = hamsterAge;
        weight = hamsterWeight;
    }
}
```



# Конструкторы

При создании объекта в теле конструктора будут инициализироваться только значения его имени, веса и возраста. Значение цвета заранее инициализировано на уровне класса — и будет всегда одним и тем же. Такое значение называется **значением поля по умолчанию**.



# Конструкторы

Присваивать значения полям на уровне класса допустимо, но старайтесь этого избегать и инициализировать их в одном месте — конструкторе. В процессе разработки код постоянно усложняется, количество полей может увеличиться в разы. Если одним будет присваиваться значение по умолчанию, а другим — в конструкторе, то возрастает риск ошибки: про какое-то из полей легко забыть и не присвоить ему никакого значения. Придётся постоянно всё перепроверять — это трудозатратно и неудобно. Лучше сразу научиться пользоваться конструктором.



# Конструкторы

- **Конструктор** - инструмент для создания новых объектов на основе класса.
- **Класс** - шаблон для создания объектов: содержит описание полей и методов.
- **Объект** - экземпляр класса. Переменная со своей собственной копией полей.

# Задача

Из-за того что мы добавили в класс **Hamster** конструктор с параметрами, код из предыдущего урока перестал работать. Перепишите его.

```
public class Hamster {
    String name;    // Имя
    int age;        // Возраст
    int weight;     // Вес в граммах
    String color;   // Цвет

    Hamster(String hamsterName, int hamsterAge, int hamsterWeight, String hamsterColor) {
        name = hamsterName;
        age = hamsterAge;
        weight = hamsterWeight;
        color = hamsterColor;
    }
}
```



```
Hamster bite = new Hamster();
bite.name = "Байт";
bite.age = 2;
bite.color = "Рыжий";
bite.weight = 350;

Hamster ninja = new Hamster();
ninja.name = "Ниндзя";
ninja.age = 1;
ninja.weight = 250;
ninja.color = "Белый";

Hamster kuzya = new Hamster();
kuzya.name = "Кузя";
kuzya.age = 3;
kuzya.weight = 450;
kuzya.color = "Пятнистый";

Hamster paskal = new Hamster();
paskal.name = "Паскаль";
paskal.age = 2;
paskal.weight = 320;
paskal.color = "Чёрный";

System.out.println(bite.name);
System.out.println(ninja.name);
System.out.println(kuzya.name);
System.out.println(paskal.name);
```

# Решение

```
public class Practice {  
    public static void main(String[] args) {  
        Hamster bite = new Hamster("Байт", 2, 350, "Рыжий");  
        Hamster ninja = new Hamster("Ниндзя", 1, 250, "Белый");  
        Hamster kuzya = new Hamster("Кузя", 3, 450, "Пятнистый");  
        Hamster paskal = new Hamster("Паскаль", 2, 320, "Чёрный");  
  
        System.out.println(bite.name);  
        System.out.println(ninja.name);  
        System.out.println(kuzya.name);  
        System.out.println(paskal.name);  
    }  
}
```



## Задача 2

Добавьте конструктор в класс для конвертера валют, который вы написали в прошлом уроке. Значения полей должны передаваться в качестве параметров со следующими названиями: **usd**, **eur**, **jpy**.

Обратите внимание: для правильной проверки вашего кода порядок параметров должен совпадать с порядком объявления полей.

```
public class Converter {  
    double rateUSD = 450;  
    double rateEUR = 500;  
    double rateJPY = 3;  
    // Добавьте конструктор здесь  
  
    void convert(double tenges, int currency) {  
        if (currency == 1) {  
            System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);  
        } else if (currency == 2) {  
            System.out.println("Ваши сбережения в евро: " + tenges / rateEUR);  
        } else if (currency == 3) {  
            System.out.println("Ваши сбережения в йенах: " + tenges / rateJPY);  
        } else {  
            System.out.println("Неизвестная валюта");  
        }  
    }  
}
```

# Решение

```
public class Converter {
    double rateUSD;
    double rateEUR;
    double rateJPY;

    public Converter(double usd, double eur, double jpy) {
        rateUSD = usd;
        rateEUR = eur;
        rateJPY = jpy;
    }

    void convert(double tenges, int currency) {
        if (currency == 1) {
            System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);
        } else if (currency == 2) {
            System.out.println("Ваши сбережения в евро: " + tenges / rateEUR);
        } else if (currency == 3) {
            System.out.println("Ваши сбережения в йенах: " + tenges / rateJPY);
        } else {
            System.out.println("Неизвестная валюта");
        }
    }
}
```

## Задача 3

Осталось исправить код в классе **Practice** — конструктор по-умолчанию больше не генерируется и возникает ошибка. При вызове конструктора с параметрами передайте следующие значения аргументов: 444.06 для доллара, 489.32 для евро и 3.81 для йены.

[https://github.com/practicetasks/java\\_tasks/tree/main/classes\\_and\\_objects/task\\_2](https://github.com/practicetasks/java_tasks/tree/main/classes_and_objects/task_2)



# Решение

<https://gist.github.com/practicetasks/305381b13e69411d6f08f4b2395b7142>



# Рефакторинг финансового приложения





# Рефакторинг финансового приложения

Ранее вы уже работали над структурой приложения — декомпозировали код на методы. Это упростило чтение и понимание программы, но работать с ней по-прежнему неудобно. Из-за того, что весь код приложения написан в одном классе **Practice** — программа выглядит громоздкой.

Это всё равно, если бы вы вдруг решили объединить тостер, кофемашину, блендер и вафельницу в один прибор. Назвали бы его — «Машина для приготовления идеального завтрака». Только вот, скорее всего, вместо «идеального завтрака» пришлось бы всё утро разбираться, как работает ваш чудо-прибор.





# Рефакторинг финансового приложения

Когда у программы появляются классы, с ней становится гораздо легче работать. Каждый из классов можно дополнять и дорабатывать отдельно. Не нужно читать и анализировать бесконечное полотно кода. В коде реальных программ и приложений могут быть тысячи классов.



# Рефакторинг финансового приложения

Вы уже знаете, что когда мы разбиваем один большой фрагмент кода на несколько смысловых блоков— это называется декомпозицией. Когда мы меняем структуру программы, в том числе добавляем в неё новые классы — это называется **рефакторингом**. Без рефакторинга и декомпозиции не написать хороший код.

В первых уроках этой темы вы уже начали рефакторинг финансового приложения — создали и описали класс **Converter**. В нём теперь хранится код, связанный с конвертацией валют:

```
public class Converter {
    double rateUSD;
    double rateEUR;
    double rateJPY;
    public Converter(double usd, double eur, double jpy) {
        rateUSD = usd;
        rateEUR = eur;
        rateJPY = jpy;
    }

    void convert(double tenges, int currency) {
        if (currency == 1) {
            System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);
        } else if (currency == 2) {
            System.out.println("Ваши сбережения в евро: " + tenges / rateEUR);
        } else if (currency == 3) {
            System.out.println("Ваши сбережения в йенах: " + tenges / rateJPY);
        } else {
            System.out.println("Неизвестная валюта");
        }
    }
}
```



# Рефакторинг финансового приложения

У класса **Converter** три поля: `rateUSD`, `rateEUR`, `rateJPY` — именно с этими валютами умеет работать финансовое приложение. В методе `convert` описана логика конвертации: выбираем валюту и конвертируем в неё остаток на счёте. С помощью параметров конструктора `Converter(double usd, double eur, double jpy)` мы можем задавать любые значения курсов валют — нам больше не надо хранить их в переменных.



# Рефакторинг финансового приложения

Теперь если мы захотим расширить возможности финансового приложения, связанные с конвертацией валют, мы будем работать сразу внутри класса **Converter**. Например, с помощью полей добавим в него новые валюты или научим конвертировать остаток на счёте сразу в несколько валют, дописав новый метод.

# Задача 1

В отдельной вкладке создайте новый класс — **DinnerAdvisor** (англ. «советник по ужину»). У него не будет полей и будет только один метод **getAdvice(double moneyBeforeSalary, int daysBeforeSalary)** — перенесите в него код из соответствующего метода класса **Practice**.

В классе **Practice** создайте объект класса **DinnerAdvisor**, воспользовавшись конструктором по умолчанию. Вызовите метод **getAdvice** класса **DinnerAdvisor** в блоке ветвления, который отвечает за реализацию пункта меню «Получить совет». Из класса **Practice** удалите метод **getAdvice(double moneyBeforeSalary, int daysBeforeSalary)**.

[https://github.com/practicetasks/java\\_tasks/tree/main/classes\\_and\\_objects/task\\_3](https://github.com/practicetasks/java_tasks/tree/main/classes_and_objects/task_3)





# Решение

<https://gist.github.com/practicetasks/f60ec6601db09ea1cd77dfeabbb435de>

## Задача 2

Завершите работу над кодом финансового приложения — соберите в отдельный класс код, касающийся трат пользователя. Назовите этот класс **ExpensesManager** (англ. «менеджер по расходам») и опишите его, действуя по пунктам:

[https://github.com/practicetasks/java\\_tasks/tree/main/classes\\_and\\_objects/task\\_4](https://github.com/practicetasks/java_tasks/tree/main/classes_and_objects/task_4)



# Решение

<https://gist.github.com/practicetasks/b26bf77cc7711c49df6ec43c08280e97>