



# Классы, наследование и взаимодействие с ними.

Урок №13

# Разбор ДЗ



10 минут



# Теория



15 минут





Опишем сущность шампуня

**Какой тип данных нам  
подойдет?**

**Объект - описание свойств  
какой-либо сущности**

# Сущность шампуня

```
1. const shampoo = {  
2.   name: 'Моё солнышко', ← Название  
3.   size: '1л',  
4.   compound: 'Вода, лауретсульфат натрия, хлорид  
   натрия',  
5. }
```


# Сущность шампуня

```
1. const shampoo = {  
2.   name: 'Моё солнышко',  
3.   size: '1л', ← Объём  
4.   compound: 'Вода, лауретсульфат натрия, хлорид  
   натрия',  
5. }
```



# Сущность шампуня

```
1. const shampoo = {  
2.   name: 'Моё солнышко',  
3.   size: '1л',  
4.   compound: 'Вода, лауретсульфат натрия, хлорид  
   натрия',  
5. }
```



**Состав**

**Что делать, если захотим создать сущность  
другого шампуня с такими же свойствами?**

## Сущность 2 шампуня

```
1.  const shampoo = {  
2.    name: 'Моё солнышко',  
3.    size: '1л',  
4.    compound: 'Вода, лауретсульфат натрия, хлорид  
        натрия',  
5.  }  
6.  const shampoo2 = {  
7.    name: 'Head and Shoulders',  
8.    size: '0.5л',  
9.    compound: 'Вода, диметикон, пиритикон цинка',  
10. }
```

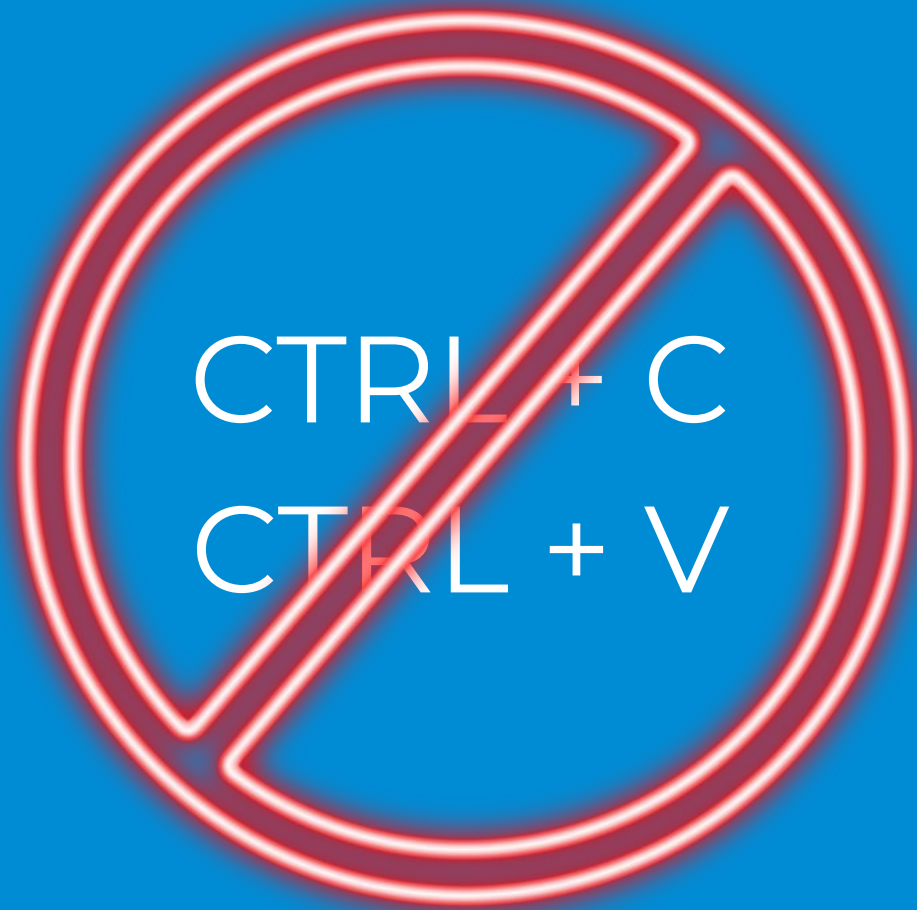
**А если сущностей будет тысячи?**



CTRL + C

CTRL + V

**Конечно есть  
выход!**





# Класс

Расширяемый шаблон кода для создания объектов, который устанавливает в них начальные значения.



# Чуть проще...

Класс - некий завод, куда мы отдаем чертёж сущности, а завод создает на основе чертежа эту сущность.





Чертёж в таких реалиях -  
свойства объекта, которые  
могут принимать разные  
значения.

# Синтаксис

```
1. class shampoo {  
2.     name = 'Моё солнышко';  
3.     size = '1л';  
4.     compound = 'Вода, лауретсульфат натрия';  
5. }
```

Классы создаются при помощи  
ключевого слова **class**



# Синтаксис

```
1. class shampoo {  
2.     name = 'Моё солнышко';  
3.     size = '1л';  
4.     compound = 'Вода, лауретсульфат натрия';  
5. }
```

После которого идет название класса (завода)!

# Синтаксис

```
1. class shampoo {  
2.     name ← 'Моё солнышко';  
3.     size = '1л';  
4.     compound = 'Вода, лауретсульфат натрия';  
5. }
```

Внутри класса могут быть переменные и функции, которые объявляются без ключевых слов, а лишь при помощи названия!

Переменные и функции класса при создании объекта превратятся в свойства и методы.


# Синтаксис

```
1. class shampoo {  
2.     name = 'Моё солнышко';  
3.     size = '1л';  
4.     compound = 'Вода, лауретсульфат натрия';  
5. }
```

**Класс** – всего-лишь описание будущего объекта, соответственно, это описание нужно превратить в фактический объект.

# Создание объекта

```
1. class Shampoo {  
2.     name = 'Моё солнышко';  
3.     size = '1л';  
4.     compound = 'Вода, лауретсульфат натрия';  
5. }  
6. let mySun = new Shampoo();
```



Ключевое слово new и  
название класса!

# Синтаксис

```
1. class Shampoo {  
2.     name = 'Моё солнышко';  
3.     size = '1л';  
4.     compound = 'Вода, лауретсульфат натрия';  
5. }  
6. let mySun = new Shampoo();
```

**mySun** – объект, созданный на основе чертежа.

# Продукт с завода

JS test.js > ...

```
1 class Shampoo {  
2   name = "Моё солнышко";  
3   size = "1л";  
4   compound = "Вода, лауретсульфат натрия";  
5 }  
6 let mySun = new Shampoo();  
7  
8 console.log(mySun)
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТЛАДКИ

ТЕРМИНАЛ

PS C:\Users\kirs\Desktop\node\классы> node test

```
Shampoo {  
  name: 'Моё солнышко',  
  size: '1л',  
  compound: 'Вода, лауретсульфат натрия'  
}
```

PS C:\Users\kirs\Desktop\node\классы> |

Объект, созданный  
на основе класса  
Shampoo!



# 3 шампуня

JS test.js > ...

```
1 class Shampoo {  
2   name = "Моё солнышко";  
3   size = "1л";  
4   compound = "Вода, лауретсульфат натрия";  
5 }  
6 let mySun1 = new Shampoo();  
7 let mySun2 = new Shampoo();  
8 let mySun3 = new Shampoo();  
9 console.log(mySun1)  
10 console.log(mySun2)  
11 console.log(mySun3)
```

Командная строка

```
C:\Users\kirs_\Desktop\node\классы>node test  
Shampoo {  
  name: 'Моё солнышко',  
  size: '1л',  
  compound: 'Вода, лауретсульфат натрия'  
}  
Shampoo {  
  name: 'Моё солнышко',  
  size: '1л',  
  compound: 'Вода, лауретсульфат натрия'  
}  
Shampoo {  
  name: 'Моё солнышко',  
  size: '1л',  
  compound: 'Вода, лауретсульфат натрия'  
}  
C:\Users\kirs_\Desktop\node\классы>
```

# Проблема

Сейчас завод создает сущность шампуня по собственному чертежу, который заложен в классе (название шампуня, состав, объем). Получается, что мы получаем их продукцию, а хотелось бы получать свою собственную.

# Конструктор

Это некий менеджер завода, который принимает техническое задание по изготовлению продукции перед тем, как её сделать.

## Контактные данные

Фамилия \*

Имя \*

Отчество \*

Мобильный телефон \* ?

E-mail \*

☐ Я соглашаюсь с условиями  
[обработки персональных данных](#)

Следующий шаг

\* отмечены поля, обязательные для заполнения



Это похоже на заполнение формы, куда мы вписываем нужные данные перед тем, как, например, зарегистрироваться на каком-нибудь сайте.



# Данные в конструктор

Функция - конструктор, выполняется перед созданием объекта. Мы будем передавать данные в конструктор, которые будут присвоены в свойства будущих объектов.

# СИНТАКСИС

```
1. class Shampoo {  
2.     constructor(name, size, compound) {  
3.         this.name = name;  
4.         this.size = size;  
5.         this.compound = compound;  
6.     }  
7. }
```

# Синтаксис

```
1. class Shampoo {  
2.     constructor(name, size, compound) {  
3.         this.name = name;  
4.         this.size = size;  
5.         this.compound = compound;  
6.     }  
7. }
```

**Функция, принимающая  
данные!**



# Синтаксис

```
1. class Shampoo {  
2.     constructor(name, size, compound) {  
3.         this.name = name;  
4.         this.size = size;  
5.         this.compound = compound;  
6.     }  
7. }
```




Принимаемые данные!



# Синтаксис

```
1. class Shampoo {  
2.     constructor(name, size, compound) {  
3.         this.name = name;  
4.         this.size = size;  
5.         this.compound = compound;  
6.     }  
7. }
```



**Присваиваем принятые  
данные свойствам  
будущих объектов!**



# Синтаксис

```
1. class Shampoo {  
2.     constructor(name, size, compound) {  
3.         this.name = name;  
4.         this.size = size;  
5.         this.compound = compound;  
6.     }  
7. }
```

**This указывает на конкретный, создаваемый объект!**

**Как передать нужные данные в  
конструктор?**

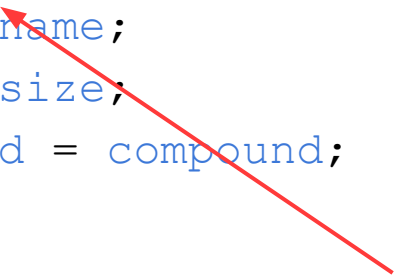
**При создании каждого  
объекта!**

# СИНТАКСИС

```
1. class Shampoo {  
2.     constructor(name, size, compound) {  
3.         this.name = name;  
4.         this.size = size;  
5.         this.compound = compound;  
6.     }  
7. }  
8. let mySha = new Shampoo("Head&Perhot", "1л", "Вода");
```


# СИНТАКСИС

```
1. class Shampoo {  
2.     constructor(name, size, compound) {  
3.         this.name = name;  
4.         this.size = size;  
5.         this.compound = compound;  
6.     }  
7. }  
8. let mySha = new Shampoo("Head&Perhot", "1л", "Вода");
```



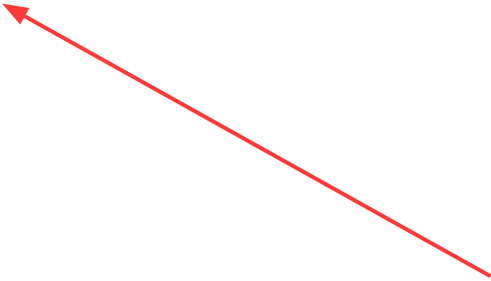
# СИНТАКСИС

```
1. class Shampoo {  
2.     constructor(name, size, compound) {  
3.         this.name = name;  
4.         this.size = size;  
5.         this.compound = compound;  
6.     }  
7. }  
8. let mySha = new Shampoo("Head&Perhot", "1л", "Вода");
```



# СИНТАКСИС


```
1. class Shampoo {  
2.     constructor(name, size, compound) {  
3.         this.name = name;  
4.         this.size = size;  
5.         this.compound = compound;  
6.     }  
7. }  
8. let mySha = new Shampoo("Head&Perhot", "1л", "Вода");
```






# СИНТАКСИС

```
1. class Shampoo {  
2.     constructor(name, size, compound) {  
3.         this.name = name;  
4.         this.size = size;  
5.         this.compound = compound;  
6.     }  
7. }  
8. let mySha = new Shampoo("Head&Perhot", "1л", "Вода");
```



# Синтаксис

```
1. class Shampoo {  
2.     constructor(name, size, compound) {  
3.         this.name = name;  
4.         this.size = size;  
5.         this.compound = compound;  
6.     }  
7. }  
8. let mySha = new Shampoo("Head&Perhot", "1л", "Вода");
```



Принятый параметр становится значением свойства конкретного, создаваемого объекта!

# Завод изготовил продукт

```
JS test.js > ...
1  class Shampoo {
2    constructor(name, size, compound) {
3      this.name = name;
4      this.size = size;
5      this.compound = compound;
6    }
7  }
8  let mySha = new Shampoo("Head&Perhot", "1л", "Вода");
9  console.log(mySha);
```

Объект шампуня на  
основе переданных  
данных!

ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ

```
PS C:\Users\kirs_\Desktop\node\классы> node test
Shampoo { name: 'Head&Perhot', size: '1л', compound: 'Вода' }
PS C:\Users\kirs_\Desktop\node\классы> |
```

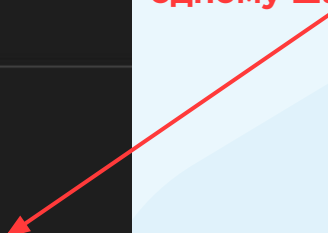
# Завод изготовил продукт

```
JS test.js > ...
1 class Shampoo {
2   constructor(name, size, compound) {
3     this.name = name;
4     this.size = size;
5     this.compound = compound;
6   }
7 }
8 console.log(new Shampoo("Head&Perhot", "1л", "Вода"));
9 console.log(new Shampoo("Крутой шампунь", "10л", "мыло"));
10 console.log(new Shampoo("Еще новый", "100г", "липтон"));
```

ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ

```
PS C:\Users\kirs_\Desktop\node\классы> node test
Shampoo { name: 'Head&Perhot', size: '1л', compound: 'Вода' }
Shampoo { name: 'Крутой шампунь', size: '10л', compound: 'мыло' }
Shampoo { name: 'Еще новый', size: '100г', compound: 'липтон' }
PS C:\Users\kirs_\Desktop\node\классы> |
```

Можно создавать  
разные объекты по  
одному шаблону!





# ОПРЕДЕЛЕНИЕ

Объект, созданный на основе класса называется **экземпляром класса**.

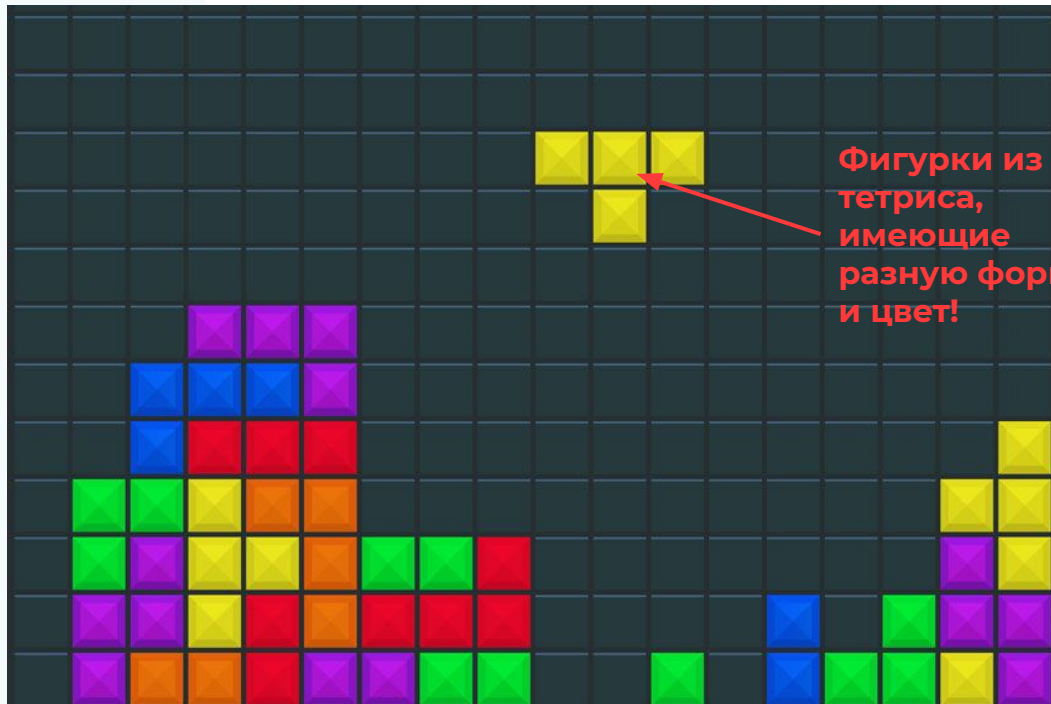
**Приведите пример объектов с одинаковыми свойствами**

# Пример



Шарики из Agar.io,  
имеющие разный  
цвет и размер!

# Пример

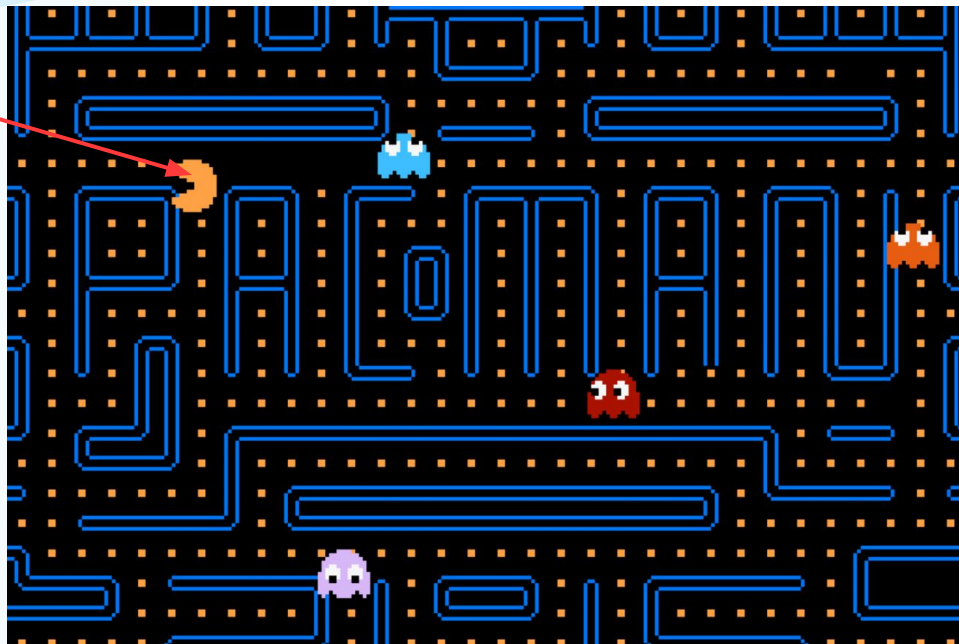


Фигурки из  
тетриса,  
имеющие  
разную форму  
и цвет!



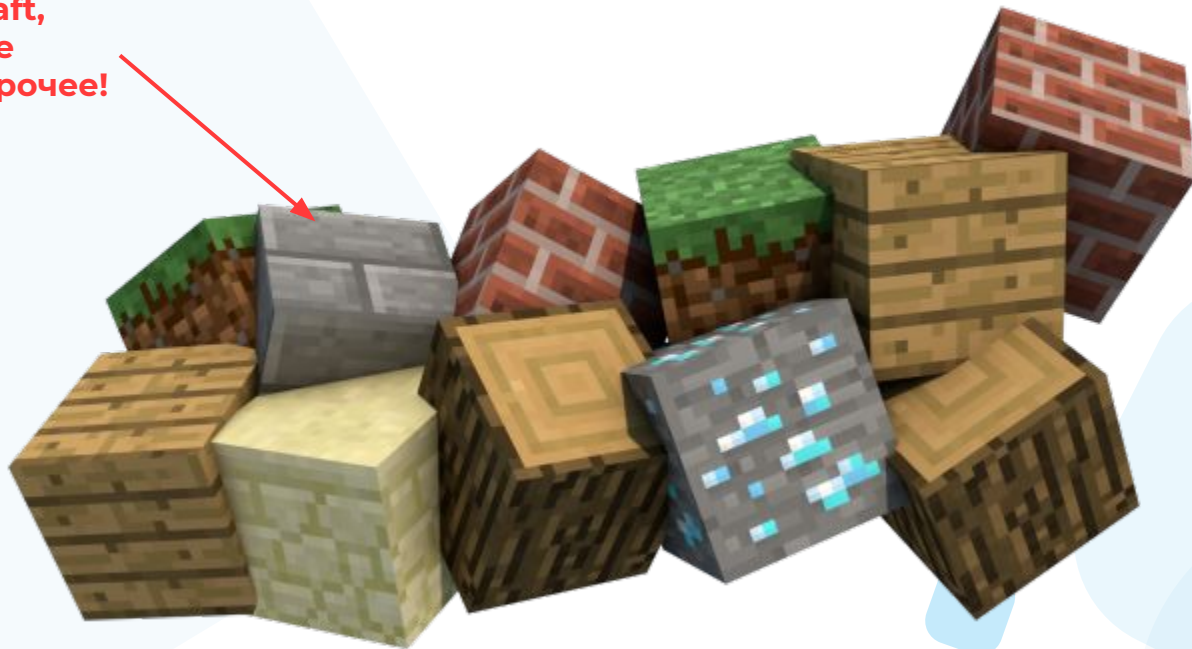
# Пример

Враги из РасМан,  
имеющие разную  
скорость и скин!



# Пример

Блоки из MineCraft,  
имеющие разные  
текстуры, ХП и прочее!



**Все эти группы объектов имеют  
одинаковый набор свойств с  
разными значениями!**

**Соответственно, под любую из этих групп можно создать класс, описывающий их свойства!**

**Какие есть вопросы?**

# Опрос

- ☐ Что такое класс?
- ☐ Для чего нужен класс?
- ☐ Как создать класс?
- ☐ Что такое экземпляр класса?
- ☐ Как создать экземпляр класса?
- ☐ Как называется функция класса, задающая значения свойствам создаваемого объекта?
- ☐ Как передать данные в функцию, срабатывающую при создании экземпляра класса?

# Конспект



```
1.  // Создание класса
2.  class Shampoo {
3.      // Функция-конструктор
4.      constructor(name, size, compound) {
5.          // Присвоение значений в свойства будущего объекта
6.          this.name = name;
7.          this.size = size;
8.          this.compound = compound;
9.      }
10. }
11. // Создание экземпляра класса с передачей данных
12. let mySha = new Shampoo("Head&Perhot", "1л", "Вода");
```

# Практика



20 минут





# Теория



15 минут



# Блок из Minecraft

```
1. class DefaultItem {
2.     // Стандартные значения для всех экземпляров
3.     width = 100;
4.     height = 100;
5.
6.     constructor(image, hp, name) {
7.         this.image = image;
8.         this.hp = hp;
9.         this.name = name;
10.    }
11. }
12. // Создание экземпляра класса
13. let stone = new DefaultItem("ссылка_на_картинку", 15,
    "Камень");
```

# Создадим много разных блоков

```
1. // Создание экземпляров класса
2. let stone = new DefaultItem("ссылка_на_картинку", 15,
    "Камень");
3. let grass = new DefaultItem("ссылка_на_картинку", 5, "Трава");
4. let dirt = new DefaultItem("ссылка_на_картинку", 3, "Земля");
5. let brick = new DefaultItem("ссылка_на_картинку", 20,
    "Кирпич");
6. let sand = new DefaultItem("ссылка_на_картинку", 2, "Песок");
```

Все они имеют разные текстуры, разное время для уничтожения и названия. Но все они связаны одинаковыми свойствами.

**А что, если мы захотим сделать блок,  
отличающийся от всех остальных...  
? например, чтобы он взрывался?**

**Нужно ли всем блокам добавлять  
возможность взрыва или проще создать  
новый класс, где будут уникальные  
свойства для определенных блоков?**

**Ни то ни другое!**



# Наследование классов

Возможность создания классов, взяв за основу все методы и свойства другого класса. Это позволяет экономить массу времени и ресурсов на написание аналогичного функционала.

# Наследование классов

```
1. class BoomItem extends DefaultItem {  
2.     constructor(????) {  
3.         ?????  
4.     }  
5. }  
6. let tnt = new BoomItem("ссылка_на_картинку", 15, "Камень");
```

Ключевое слово для наследования!



# Наследование классов

```
1. class BoomItem extends DefaultItem {  
2.     constructor(????) {  
3.         ?????  
4.     }  
5. }  
6. let tnt = new BoomItem("ссылка_на_картинку", 15, "Камень");
```

Название класса, у  
которого скопируем все  
свойства и методы!

# Наследование классов


```
1. class BoomItem extends DefaultItem {  
2.     constructor(????) {  
3.         ?????  
4.     }  
5. }  
6. let tnt = new BoomItem("ссылка_на_картинку", 15, "Камень");
```

Что в конструкторе?

**Данные также необходимо принять!**

# Наследование классов

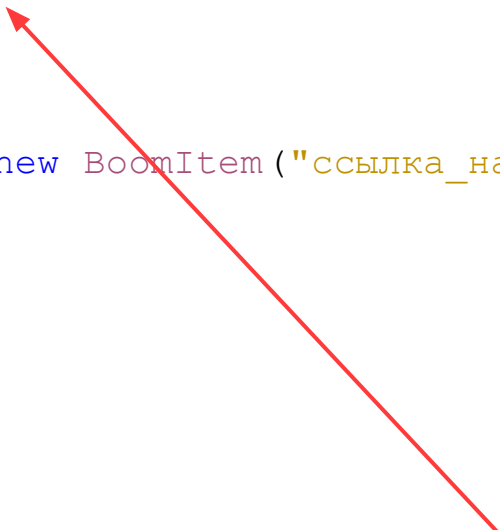
```
1. class BoomItem extends DefaultItem {  
2.     constructor(image, hp, name) {  
3.         ?????  
4.     }  
5. }  
6. let tnt = new BoomItem("ссылка_на_картинку", 15, "Камень");
```



**Принимаем!**

# Наследование классов

```
1. class BoomItem extends DefaultItem {  
2.     constructor(image, hp, name) {  
3.         ?????  
4.     }  
5. }  
6. let tnt = new BoomItem("ссылка_на_картинку", 15, "Камень");
```



А здесь снова  
присваивать текущему  
объекту?

**Нет!**

# Наследование классов

```
1. class BoomItem extends DefaultItem {  
2.     constructor(image, hp, name) {  
3.         super(image, hp, name);  
4.     }  
5. }  
6. let tnt = new BoomItem("ссылка_на_картинку", 15, "Камень");
```

Функция super()!

# Функция `super()`

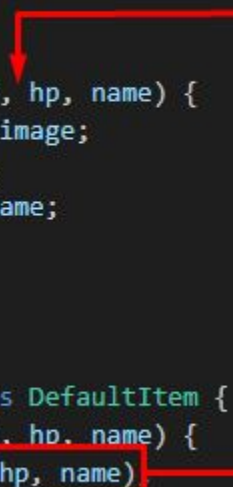
Вызывает функцию-конструктор класса-родителя и передает туда полученные свойства



# Передача данных в родительский конструктор!

JS test.js > ...

```
1 class DefaultItem {  
2   width = 100;  
3   height = 100;  
4  
5  
6   constructor(image, hp, name) {  
7     this.image = image;  
8     this.hp = hp;  
9     this.name = name;  
10  }  
11 }  
12  
13  
14 class BoomItem extends DefaultItem {  
15   constructor(image, hp, name) {  
16     super(image, hp, name)  
17   }  
18 }  
19
```



A red box highlights the `super(image, hp, name)` call in the `BoomItem` constructor. A red arrow points from this call to the `constructor` method of the `DefaultItem` class, illustrating the flow of data from the child class to the parent class.

# Расширение функционала

```
1. class BoomItem extends DefaultItem {  
2.     constructor(image, hp, name, timeToBoom) {  
3.         super(image, hp, name);  
4.  
5.         this.time = timeToBoom;  
6.         this.boom = true;  
7.     }  
8. }  
9. let tnt = new BoomItem("ссылка_на_картинку", 15, "Камень", 5);
```



**Наследуемся от  
нового класса!**


# Расширение функционала

```
1. class BoomItem extends DefaultItem {  
2.     constructor(image, hp, name, timeToBoom) {  
3.         super(image, hp, name);  
4.  
5.         this.time = timeToBoom;  
6.         this.boom = true;  
7.     }  
8. }  
9. let tnt = new BoomItem("ссылка_на_картинку", 15, "Камень", 5);
```



Принимаем новые  
параметры!


# Расширение функционала

```
1. class BoomItem extends DefaultItem {  
2.     constructor(image, hp, name, timeToBoom) {  
3.         super(image, hp, name);  
4.  
5.         this.time = timeToBoom;  Добавляем новые  
свойства объекта!  
6.         this.boom = true;  
7.     }  
8. }  
9. let tnt = new BoomItem("ссылка_на_картинку", 15, "Камень", 5);
```

# Расширение функционала

```
1. class BoomItem extends DefaultItem {  
2.     constructor(image, hp, name, timeToBoom) {  
3.         super(image, hp, name);  
4.  
5.         this.time = timeToBoom;  
6.         this.boom = true;  
7.     }  
8. }  
9. let tnt = new BoomItem("ссылка_на_картинку", 15, "Камень", 5);
```

Не забываем  
передавать новый  
аргумент!



# Работает!

```
JS test.js > BoomItem > constructor
1 class DefaultItem {
2   width = 100;
3   height = 100;
4   constructor(image, hp, name) {
5     this.image = image;
6     this.hp = hp;
7     this.name = name;
8   }
9 }
10 class BoomItem extends DefaultItem {
11   constructor(image, hp, name, timeToBoom) {
12     super(image, hp, name);
13     this.time = timeToBoom;
14     this.boom = true;
15   }
16 }
17 let tnt = new BoomItem("ссылка_на_картинку", 1, "Tnt", 5);
18 console.log(tnt);
```

ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ

```
PS C:\Users\kirs\Desktop\node\классы> node test
BoomItem {
  width: 100,
  height: 100,
  image: 'ссылка_на_картинку',
  hp: 1,
  name: 'Tnt',
  time: 5,
  boom: true
}
PS C:\Users\kirs\Desktop\node\классы> |
```

**Вопросики?**

# Конспект



```
1.  class DefaultItem {
2.      width = 100;
3.      height = 100;
4.      constructor(image, hp, name) {
5.          this.image = image;
6.          this.hp = hp;
7.          this.name = name;
8.      }
9.  }
10. // Унаследованный класс от DefaultItem
11. class BoomItem extends DefaultItem {
12.     constructor(image, hp, name, timeToBoom) {
13.         // Вызывает конструктор родительского класса, передавая туда данные
14.         super(image, hp, name);
15.         // Дополнительные свойства
16.         this.boom = true;
17.         this.time = timeToBoom;
18.     }
19. }
```



# Практика



20 минут



# Итоги урока

- 1) Узнали что такое класс
- 2) Научились создавать классы
- 3) Узнали про функцию-конструктор
- 4) Научились наследовать классы
- 5) Узнали про функцию `super()`
- 6) Познакомились с ООП в JavaScript