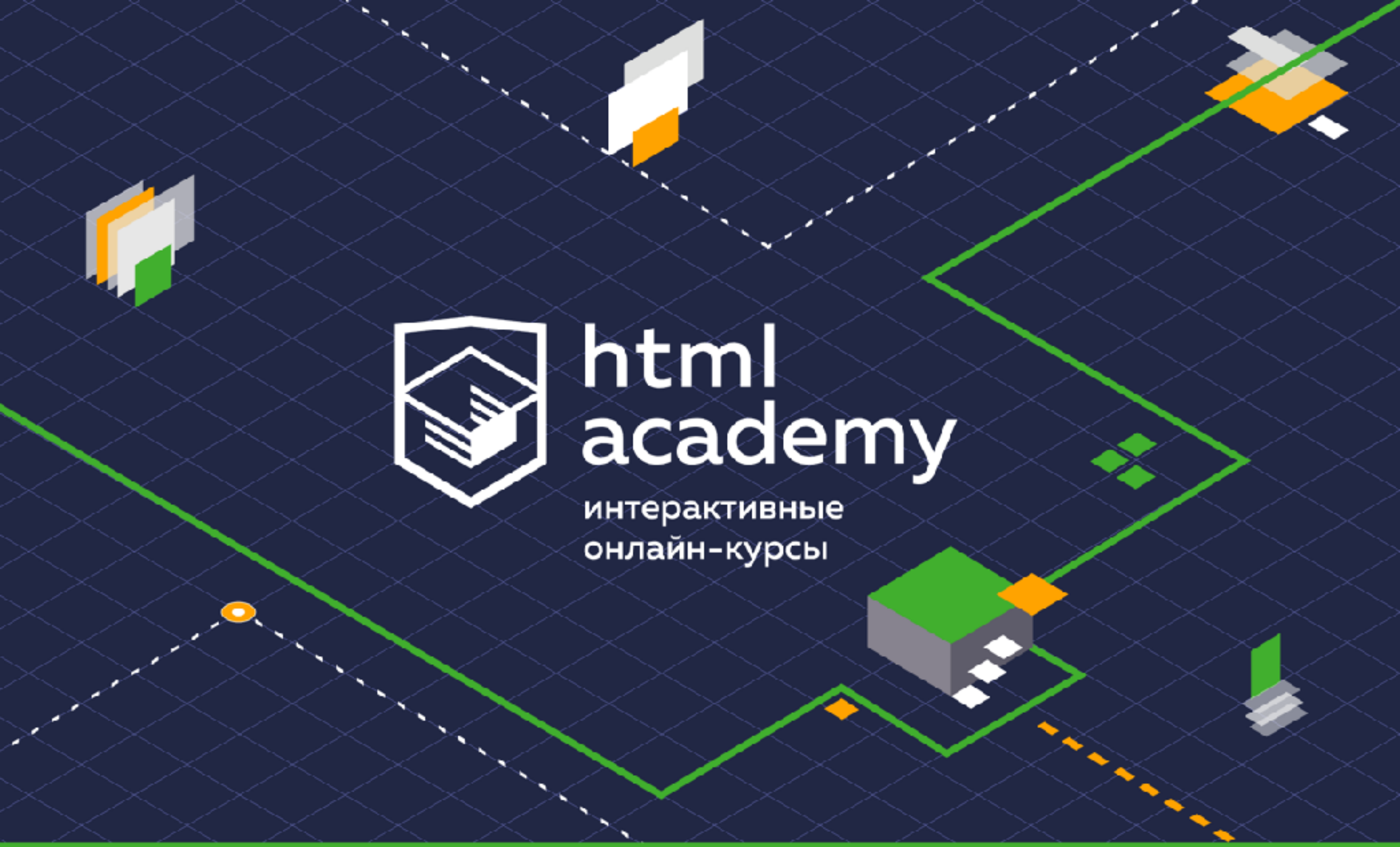


0x05 раздел: Компонентный подход

В этой серии

- Новые возможности объектов
- Классы
- ООП
- Паттерны проектирования
- UML

Объекты

Объект в ES5

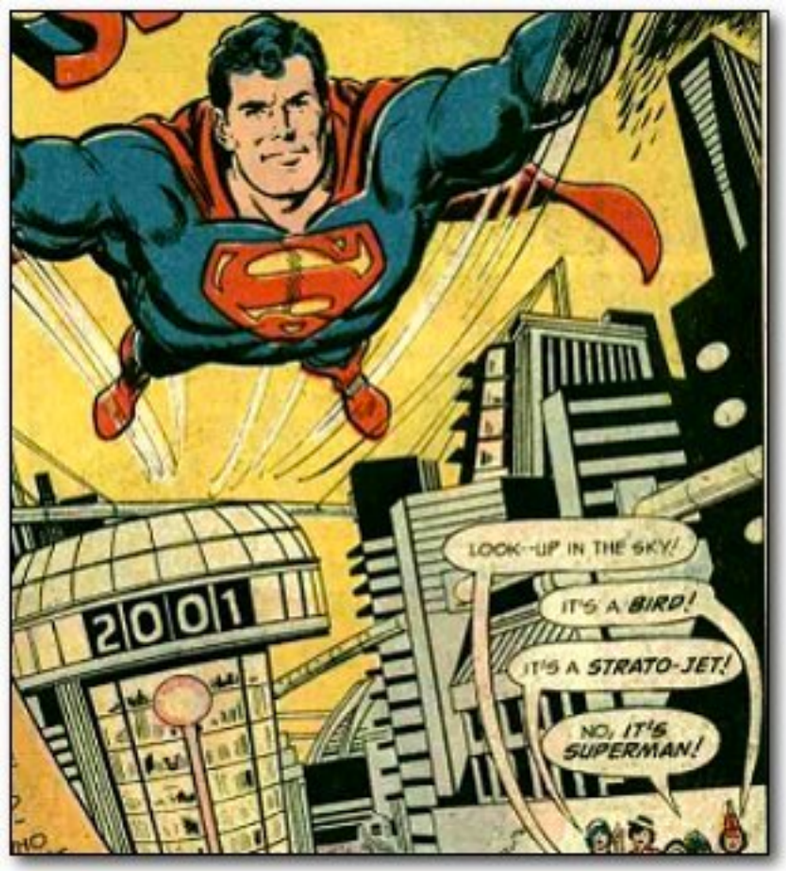
```
const name = 'Барсик';
const age = 4;
const male = true;

const cat = {
  name: name,
  age: age,
  male: male
};
console.dir(cat); // { name: 'Барсик', age: 4, male: true }
```

Объект в ES2015

```
const name = 'Барсик';
const age = 4;
const male = true;

const cat = {
  name,
  age,
  male
};
console.dir(cat); // { name: 'Барсик', age: 4, male: true }
```

Что описывает этот объект?

```
const name = 'Барсик';
const age = 4;
const male = true;

// Кого описываем сегодня?
const dog = { name, age, male }; // Это пес?!
const cat = { name, age, male }; // Это кот?!
const ship = { name, age, male };// Это человек и пароход?!
```



Добавим метод ES5

```
const cat = {
  name: 'Барсик',
  age: 4,
  meow: function () {
    return 'Мяу!';
  }
};

console.log(cat.meow()); // "Мяу!"
```

Добавим метод ES2015

```
const cat = {
  name: 'Барсик',
  age: 4,
  meow() {
    return 'Мяу!';
  }
};
```

```
console.log(cat.meow()); // "Мяу!"
```

Отличим по поведению

```
const keks = {
  name: 'Keks',
  meow() {
    return 'Мяо!';
  }
};

const bob = {
  name: 'Bobik',
  woof() {
    return 'Гаф!';
  }
};

const isCat = (obj) => {
  return (typeof (obj.meow) === 'function') && obj.name;
};

console.log(isCat(keks)); // true
console.log(isCat(bob)); // false
```

Утиная типизация (*Duck typing*)

Если это выглядит как утка, плавает как утка и крякает как утка, то это, возможно, и есть утка.

en.wikipedia.org/wiki/Duck_typing

Утиная типизация (*Duck typing*)

```
const keks = {
  name: 'Keks',
  meow() {
    return 'Мяо!';
  }
};

const vasilyIvanovich = {
  name: 'Василий Иванович',
  meow() {
    return 'Мяу, чё!';
  }
};

const isCat = (obj) => {
  return obj.name && (typeof (obj.meow) === 'function');
};

console.log(isCat(keks));              // true
console.log(isCat(vasilyIvanovich));  // true
```

Конструктор

```
const CAT_TYPE = `Котэ`;

const newCat = (name, male = true) => {
  return {
    kind: CAT_TYPE,
    name,
    male,
    meow() {
      return 'Мяу!';
    }
  }
};

const keks = newCat('Keks');
console.log(keks);
// { kind: 'Котэ', name: 'Keks', male: true, meow: [Function: meow] }
console.log(keks.kind);// Котэ

const catInBag = {
  name: 'Кеша',
  meow() {
    return 'Мяу! Кеша попугай!';
  }
};
console.log(catInBag.kind); // undefined
```

Функция-конструктор

```
const Cat = function (name, male = true) {
  this.name = name;
  this.male = male;
};

Cat.prototype.meow = function () {
  return 'Мяу!';
};
```

```
const keks = new Cat('Keks');
console.log(keks);                    // { name: 'Keks', male: true }
console.log(keks.meow());             // Мяу!
console.log(keks instanceof Cat);     // true

const catInBag = {
  name: 'Кеша',
  meow() {
    return 'Мяу! Кеша попугай!';
  }
};
console.log(catInBag instanceof Cat); // false
```

ООП

Объектно-ориентированное программирование

Класс

группа предметов или явлений, обладающих общими признаками

Класс описывает

- Что это такое?
- Как оно взаимодействует с другими объектами?
- Как другие объекты могут взаимодействовать с этим?
- Как создать новый элемент?

Класс ES2015

```
class Cat {
  constructor(name, age = 0, male = true) {
    this.name = name;
    this.age = age;
    this.male = male;
  }

  meow() {
    return 'Мяу!';
  }
}

const keksCat = new Cat('Кекс', 4);
const snezhinkaCat = new Cat('Снежинка', 3, false);

keksCat.meow();
snezhinkaCat.meow();
```

ООП

- Абстракция
  обобщение данных

- Инкапсуляция
  изоляция данных, поведения

- Наследование
  дополнение и уточнение данных

- Полиморфизм
  изменение поведения при наследовании

Абстракция данных

```
class Cat {
  constructor(name, age = 0, male = true) {
    this.name = name;
    this.age = age;
    this.male = male;
  }

  say() {
    return 'Мяу!';
  }
}

const cat = new Cat('Grumpy', 3);
console.info(cat.say()); // Мяу!

class Thing {
}

console.info(new Thing()); // Thing {}
```

Инкапсуляция

```
class Cat {
  constructor(name, speed = 3, age = 1) {
    this.name = name;
    this.speed = speed;
    this.age = age;

    this._totalDistance = 0;
  }

  run(time) {
    const distance = time * this.speed;
    console.log(`${this.name} run ${distance}`);
    this._totalDistance += distance;
  }

  getPerformance() {
    return this._totalDistance / this.age;
  }
}
```

Наследование (Родитель)

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  greet() {
    console.log(`I'm ${this.name}. Nice to meet you!`);
  }
}

export default Animal;
```

Наследование (Потомок)

```
import Animal from './09-inherit-parent';

class Cat extends Animal {
  constructor(name, age) {
    super(name);
    this.age = age;
  }

  meow() {
    console.log('Мяу!');
  }
}

class Dog extends Animal {
  constructor(name, age) {
    super(name);
    this.age = age;
  }

  bark() {
    console.log('Гав!');
  }
}

const cat = new Cat('Keks', 5);
cat.greet();// I'm Keks. Nice to meet you!
cat.meow(); // Мяу!
cat.bark(); // Ошибка!

const dog = new Dog('Bim', 2);
dog.greet();// I'm Bim. Nice to meet you!
dog.bark(); // Гав!
dog.meow(); // Ошибка!
```

Полиморфизм (Родитель)

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  say() {
    // What I have to say and how?
  }

  toString() {
    console.log(`I'm an animal with name: ${this.name}`);
  }
}

export default Animal;
```

Полиморфизм (Потомок)

```
import Animal from './09-inheritance-parent';

class Cat extends Animal {
  constructor(name, age) {
    super(name);
    this.age = age;
  }

  say() {
    console.log('Мяу!');
  }
}

class Dog extends Animal {
  constructor(name, age) {
    super(name);
    this.age = age;
  }

  say() {
    console.log('Гав!');
  }
}

const cat = new Cat('Keks', 5);
cat.say(); // Мяу!
const dog = new Dog('Bim', 2);
dog.say(); // Гав!
```

Переопределение (override)

```
import Animal from './10-polymorph-parent';

class Cat extends Animal {
  constructor(name, age) {
    super(name);
    this.age = age;
  }

  say() {
    console.log('Мяу!');
  }
}

class Kitten extends Cat {
  constructor(name, age) {
    super(name, age);
  }

  say() {
    if (this.age > 1) {
      super.say();
    } else {
      console.log('Миу!');
    }
  }
}

new Kitten('Гав', 1).say(); // Миу!
new Kitten('Гав', 3).say(); // Мяу!
```

Прототипное наследование (ES5)

```
var Shape = function (id, x, y) {
  this.id = id;
  this.move(x, y);
};

Shape.prototype.move = function (x, y) {
  this.x = x;
  this.y = y;
};
```

```
var Rectangle = function (id, x, y, width, height) {
  Shape.call(this, id, x, y);
  this.width  = width;
  this.height = height;
};
Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;

var Circle = function (id, x, y, radius) {
  Shape.call(this, id, x, y);
  this.radius = radius;
};
Circle.prototype = Object.create(Shape.prototype);
Circle.prototype.constructor = Circle;
```

Классы ES2015

```
class Shape {
  constructor (id, x, y) {
    this.id = id;
    this.move(x, y);
  }
  move (x, y) {
    this.x = x;
    this.y = y;
  }
}
```

```
class Rectangle extends Shape {
  constructor (id, x, y, width, height) {
    super(id, x, y);
    this.width  = width;
    this.height = height;
  }
}
class Circle extends Shape {
  constructor (id, x, y, radius) {
    super(id, x, y);
    this.radius = radius;
  }
}
```

Доступ к базовому классу

```
class Shape {
    …
    toString () {
      return `Shape(${this.id})`;
    }
}
class Rectangle extends Shape {
  constructor (id, x, y, width, height) {
    super(id, x, y);
        …
  }
  toString () {
    return "Rectangle > " +
      super.toString(); // Shape.prototype.toString.call(this)
  }
}
class Circle extends Shape {
  constructor (id, x, y, radius) {
    super(id, x, y);
        …
  }
  toString () {
    return "Circle > " +
      super.toString(); // Shape.prototype.toString.call(this)
  }
}
```

Статические методы

```
import assert from 'assert';

class Rectangle extends Shape {
  static fitCircle(circle) {
    const side = circle.radius;
    return new Rectangle(circle.id, circle.x, circle.y, side, side);
  }
}
class Circle extends Shape {
  static fitRectangle(rect) {
    const minSide = Math.min(rect.height, rect.width);
    return new Circle(rect.id, rect.x, rect.y, minSide);
  }
}


assert.deepEqual(Circle.fitRectangle(new Rectangle("1", 0, 0, 100, 200)),
  new Circle("1", 0, 0, 100));
assert.deepEqual(Rectangle.fitCircle(new Circle("0", 0, 0, 50)),
  new Rectangle("0", 0, 0, 50, 50));
```

Проектирование

Как построить дом?

Обычный дом

- Построить первый этаж
- Построить второй этаж
- Построить третий этаж
- Построить четвертый этаж
- Построить пятый этаж

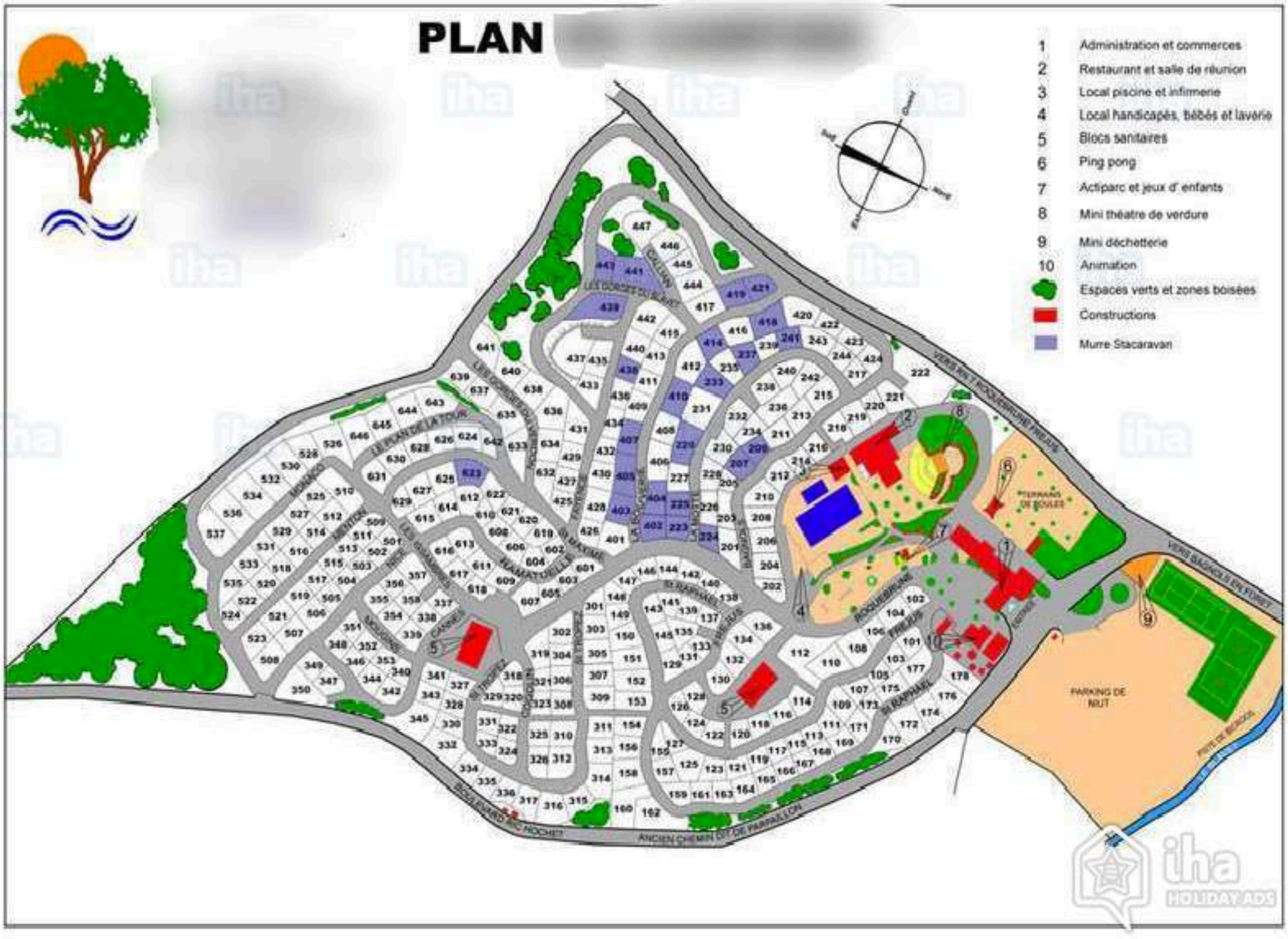
Рекурсивный дом

- Взять одноэтажный дом
- Пристроить один этаж сверху — двухэтажный дом
- Пристроить один этаж сверху — трехэтажный дом
- Пристроить один этаж сверху — четырехэтажный дом
- Пристроить один этаж сверху — пятиэтажный дом

ООП дом

- Создать класс, который описывает Дом — проект дома
- Создать новый объект Дом с параметром 2 — двухэтажный дом
- Создать новый объект Дом с параметром 3 — трехэтажный дом
- Создать новый объект Дом с параметром 4 — четырехэтажный дом
- Создать новый объект Дом с параметром 5 — пятиэтажный дом

Как построить квартал?



Паттерны проектирования

повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста

ru.wikipedia.org/wiki/Шаблон_проектирования

Паттерны проектирования

- Фундаментальные
  стандартные подходы к программированию

- Порождающие
  создают новые объекты

- Структурные
  описывают стандартные структуры данных

- Поведенческие
  стандартные способы, описать поведение приложения

- Анти-паттерны
  как делать не стоило =(

habrahabr.ru/post/136766

Знакомо?

```
const link = document.querySelector('a');

link.addEventListener('click', (evt) => {
  evt.preventDefault();

  console.info('Клик по ссылке =)');
});
```

Паттерн Слушатель — поведенческий

```
const link = document.querySelector('a');

link.addEventListener('click', (evt) => {
  evt.preventDefault();

  console.info('Клик по ссылке =)');
});
```

Знакомо?

```
const newCat = (name, male = true) => {
  return {
    kind: CAT_TYPE,
    name,
    male,
    meow() {
      return 'Мяу!';
    }
  }
};
```

Паттерн Фабричный метод — порождающий

```
const newCat = (name, male = true) => {
  return {
    kind: CAT_TYPE,
    name,
    male,
    meow() {
      return 'Мяу!';
    }
  };
```

Знакомо?

```
import {createElement, changeView} from './util';
import renderFooter from './game/footer';
import renderHeader from './game/header';
import renderLevel from './game/game-level';
import {initialGame, setCurrentLevel, setTime, getLevel} from './data/quest';
import {Result} from './data/quest-data';

import end from './end';
import die from './game/death-screen';


let gameState = initialGame;
let interval = null;

const root = createElement(``);
const headerElement = renderHeader(gameState);
const levelElement = renderLevel(getLevel(gameState.level));
const footerElement = renderFooter();

root.appendChild(headerElement);
root.appendChild(levelElement);
root.appendChild(footerElement);

const updateHeader = (game) => {
  renderHeader(game);
};

const onAnswer = (answer) => {
  switch (answer.result) {
    case Result.DIE:
      onExit();
      die(gameState);
      break;
    case Result.NEXT:
      gameState = setCurrentLevel(gameState, gameState.level + 1);
      update();
      break;
    case Result.WIN:
      onExit();
      end();
      break;
    default:
      throw new Error(`Unknown result: ${answer.result}`);
  }
};

const updateLevel = (level) => {
  const gameElement = renderLevel(level);

  const answers = gameElement.querySelector(`ul.answers`).querySelectorAll(`li`);

  for (let i = 0; i < answers.length; i++) {
    const answer = answers[i];
    answer.onclick = (evt) => {
      evt.preventDefault();

      onAnswer(level.answers[i]);
    };
  }

  const answerInput = gameElement.querySelector(`input`);

  answerInput.onkeydown = (evt) => {
    if (evt.keyCode === 13) {

      const text = answerInput.value;
      const answer = level.answers.find((it) => it.action.toLowerCase().startsWith(text));

      if (answer) {
        onAnswer(answer);
      } else {
        onAnswer(text);
      }
      answerInput.value = ``;
    }
  };

  answerInput.focus();
};

const update = () => {
  updateHeader(gameState);
  updateLevel(getLevel(gameState.level));
  // Footer never changes, so never update
};

const onExit = () => {
  clearInterval(interval);
};

const onEnter = () => {
  update();

  interval = setInterval(() => {
```

```
const updateHeader = (game) => {
  renderHeader(game);
};

const onAnswer = (answer) => {
  switch (answer.result) {
    case Result.DIE:
      onExit();
      die(gameState);
      break;
    case Result.NEXT:
      gameState = setCurrentLevel(gameState, gameState.level + 1);
      update();
      break;
    case Result.WIN:
      onExit();
      end();
      break;
    default:
      throw new Error(`Unknown result: ${answer.result}`);
  }
};

const updateLevel = (level) => {
  const gameElement = renderLevel(level);

  const answers = gameElement.querySelector(`ul.answers`).querySelectorAll(`li`);

  for (let i = 0; i < answers.length; i++) {
    const answer = answers[i];
    answer.onclick = (evt) => {
      evt.preventDefault();

      onAnswer(level.answers[i]);
    };
  }

  const answerInput = gameElement.querySelector(`input`);

  answerInput.onkeydown = (evt) => {
    if (evt.keyCode === 13) {

      const text = answerInput.value;
      const answer = level.answers.find((it) => it.action.toLowerCase().startsWith(text));

      if (answer) {
        onAnswer(answer);
      } else {
        onAnswer(text);
      }
      answerInput.value = ``;
    }
  };

  answerInput.focus();
};

const update = () => {
  updateHeader(gameState);
  updateLevel(getLevel(gameState.level));
  // Footer never changes, so never update
};

const onExit = () => {
  clearInterval(interval);
};

const onEnter = () => {
  update();

  interval = setInterval(() => {
    gameState = setTime(gameState, gameState.time + 1);
    updateHeader(gameState);
  }, 1000);

  changeView(root);
};

export default (state = initialGame) => {
  gameState = state;

  onEnter();
};

const updateLevel = (level) => {
  const gameElement = renderLevel(level);

  const answers = gameElement.querySelector(`ul.answers`).querySelectorAll(`li`);

  for (let i = 0; i < answers.length; i++) {
    const answer = answers[i];
    answer.onclick = (evt) => {
      evt.preventDefault();
```

```
const updateLevel = (level) => {
  const gameElement = renderLevel(level);

  const answers = gameElement.querySelector(`ul.answers`).querySelectorAll(`li`);

  for (let i = 0; i < answers.length; i++) {
    const answer = answers[i];
    answer.onclick = (evt) => {
      evt.preventDefault();

      onAnswer(level.answers[i]);
    };
  }

  const answerInput = gameElement.querySelector(`input`);

  answerInput.onkeydown = (evt) => {
    if (evt.keyCode === 13) {

      const text = answerInput.value;
      const answer = level.answers.find((it) => it.action.toLowerCase().startsWith(text));

      if (answer) {
        onAnswer(answer);
      } else {
        onAnswer(text);
      }
      answerInput.value = ``;
    }
  };

  answerInput.focus();
};

const update = () => {
  updateHeader(gameState);
  updateLevel(getLevel(gameState.level));
  // Footer never changes, so never update
};

const onExit = () => {
  clearInterval(interval);
};

const root = createElement(``);
const headerElement = renderHeader(gameState);
const levelElement = renderLevel(getLevel(gameState.level));
const footerElement = renderFooter();

root.appendChild(headerElement);
root.appendChild(levelElement);
root.appendChild(footerElement);

const updateHeader = (game) => {
  renderHeader(game);
};

const onAnswer = (answer) => {
  switch (answer.result) {
    case Result.DIE:
      onExit();
      die(gameState);
      break;
    case Result.NEXT:
      gameState = setCurrentLevel(gameState, gameState.level + 1);
      update();
      break;
    case Result.WIN:
      onExit();
      end();
      break;
    default:
      throw new Error(`Unknown result: ${answer.result}`);
  }
};

const updateLevel = (level) => {
  const gameElement = renderLevel(level);

  const answers = gameElement.querySelector(`ul.answers`).querySelectorAll(`li`);

  for (let i = 0; i < answers.length; i++) {
    const answer = answers[i];
    answer.onclick = (evt) => {
      evt.preventDefault();

      onAnswer(level.answers[i]);
    };
  }

  const answerInput = gameElement.querySelector(`input`);

  answerInput.onkeydown = (evt) => {
    if (evt.keyCode === 13) {

      const text = answerInput.value;
```

Антипаттерн Спагетти-код (Spaghetti code)

```
import {createElement, changeView} from './util';
import renderFooter from './game/footer';
import renderHeader from './game/header';
import renderLevel from './game/game-level';
import {initialGame, setCurrentLevel, setTime, getLevel} from './data/quest';
import {Result} from './data/quest-data';

import end from './end';
import die from './game/death-screen';


let gameState = initialGame;
let interval = null;

const root = createElement(``);
const headerElement = renderHeader(gameState);
const levelElement = renderLevel(getLevel(gameState.level));
const footerElement = renderFooter();

root.appendChild(headerElement);
root.appendChild(levelElement);
root.appendChild(footerElement);

const updateHeader = (game) => {
  renderHeader(game);
};

const onAnswer = (answer) => {
  switch (answer.result) {
    case Result.DIE:
      onExit();
      die(gameState);
      break;
    case Result.NEXT:
      gameState = setCurrentLevel(gameState, gameState.level + 1);
      update();
      break;
    case Result.WIN:
      onExit();
      end();
      break;
    default:
      throw new Error(`Unknown result: ${answer.result}`);
  }
};

const updateLevel = (level) => {
  const gameElement = renderLevel(level);

  const answers = gameElement.querySelector(`ul.answers`).querySelectorAll(`li`);

  for (let i = 0; i < answers.length; i++) {
    const answer = answers[i];
    answer.onclick = (evt) => {
      evt.preventDefault();

      onAnswer(level.answers[i]);
    };
  }

  const answerInput = gameElement.querySelector(`input`);

  answerInput.onkeydown = (evt) => {
    if (evt.keyCode === 13) {

      const text = answerInput.value;
      const answer = level.answers.find((it) => it.action.toLowerCase().startsWith(text));

      if (answer) {
        onAnswer(answer);
      } else {
        onAnswer(text);
      }
      answerInput.value = ``;
    }
  };

  answerInput.focus();
};

const update = () => {
  updateHeader(gameState);
  updateLevel(getLevel(gameState.level));
  // Footer never changes, so never update
};

const onExit = () => {
  clearInterval(interval);
};

const onEnter = () => {
  update();

  interval = setInterval(() => {
```

```
const updateHeader = (game) => {
  renderHeader(game);
};

const onAnswer = (answer) => {
  switch (answer.result) {
    case Result.DIE:
      onExit();
      die(gameState);
      break;
    case Result.NEXT:
      gameState = setCurrentLevel(gameState, gameState.level + 1);
      update();
      break;
    case Result.WIN:
      onExit();
      end();
      break;
    default:
      throw new Error(`Unknown result: ${answer.result}`);
  }
};

const updateLevel = (level) => {
  const gameElement = renderLevel(level);

  const answers = gameElement.querySelector(`ul.answers`).querySelectorAll(`li`);

  for (let i = 0; i < answers.length; i++) {
    const answer = answers[i];
    answer.onclick = (evt) => {
      evt.preventDefault();

      onAnswer(level.answers[i]);
    };
  }

  const answerInput = gameElement.querySelector(`input`);

  answerInput.onkeydown = (evt) => {
    if (evt.keyCode === 13) {

      const text = answerInput.value;
      const answer = level.answers.find((it) => it.action.toLowerCase().startsWith(text));

      if (answer) {
        onAnswer(answer);
      } else {
        onAnswer(text);
      }
      answerInput.value = ``;
    }
  };

  answerInput.focus();
};

const update = () => {
  updateHeader(gameState);
  updateLevel(getLevel(gameState.level));
  // Footer never changes, so never update
};

const onExit = () => {
  clearInterval(interval);
};

const onEnter = () => {
  update();

  interval = setInterval(() => {
    gameState = setTime(gameState, gameState.time + 1);
    updateHeader(gameState);
  }, 1000);

  changeView(root);
};

export default (state = initialGame) => {
  gameState = state;

  onEnter();
};

const updateLevel = (level) => {
  const gameElement = renderLevel(level);

  const answers = gameElement.querySelector(`ul.answers`).querySelectorAll(`li`);

  for (let i = 0; i < answers.length; i++) {
    const answer = answers[i];
    answer.onclick = (evt) => {
      evt.preventDefault();
```

```
const updateLevel = (level) => {
  const gameElement = renderLevel(level);

  const answers = gameElement.querySelector(`ul.answers`).querySelectorAll(`li`);

  for (let i = 0; i < answers.length; i++) {
    const answer = answers[i];
    answer.onclick = (evt) => {
      evt.preventDefault();

      onAnswer(level.answers[i]);
    };
  }

  const answerInput = gameElement.querySelector(`input`);

  answerInput.onkeydown = (evt) => {
    if (evt.keyCode === 13) {

      const text = answerInput.value;
      const answer = level.answers.find((it) => it.action.toLowerCase().startsWith(text));

      if (answer) {
        onAnswer(answer);
      } else {
        onAnswer(text);
      }
      answerInput.value = ``;
    }
  };

  answerInput.focus();
};

const update = () => {
  updateHeader(gameState);
  updateLevel(getLevel(gameState.level));
  // Footer never changes, so never update
};

const onExit = () => {
  clearInterval(interval);
};

const root = createElement(``);
const headerElement = renderHeader(gameState);
const levelElement = renderLevel(getLevel(gameState.level));
const footerElement = renderFooter();

root.appendChild(headerElement);
root.appendChild(levelElement);
root.appendChild(footerElement);

const updateHeader = (game) => {
  renderHeader(game);
};

const onAnswer = (answer) => {
  switch (answer.result) {
    case Result.DIE:
      onExit();
      die(gameState);
      break;
    case Result.NEXT:
      gameState = setCurrentLevel(gameState, gameState.level + 1);
      update();
      break;
    case Result.WIN:
      onExit();
      end();
      break;
    default:
      throw new Error(`Unknown result: ${answer.result}`);
  }
};

const updateLevel = (level) => {
  const gameElement = renderLevel(level);

  const answers = gameElement.querySelector(`ul.answers`).querySelectorAll(`li`);

  for (let i = 0; i < answers.length; i++) {
    const answer = answers[i];
    answer.onclick = (evt) => {
      evt.preventDefault();

      onAnswer(level.answers[i]);
    };
  }

  const answerInput = gameElement.querySelector(`input`);

  answerInput.onkeydown = (evt) => {
    if (evt.keyCode === 13) {

      const text = answerInput.value;
```

Антипаттерн Магические числа (Magic numbers)

```
switch (answer.result) {
  case 0:
    onExit();
    die(gameState);
    break;
  case 1:
    gameState = setCurrentLevel(gameState, gameState.level + 1);
    update();
    break;
  case 3:
    onExit();
    end();
    break;
  default:
    throw new Error(`Unknown result: ${answer.result}`);
}
```

Паттерн Перечисление (Enumerable)

```
export const Result = {
  DIE: 0,
  NOOP: 1,
  NEXT: 2,
  WIN: 3
};
```

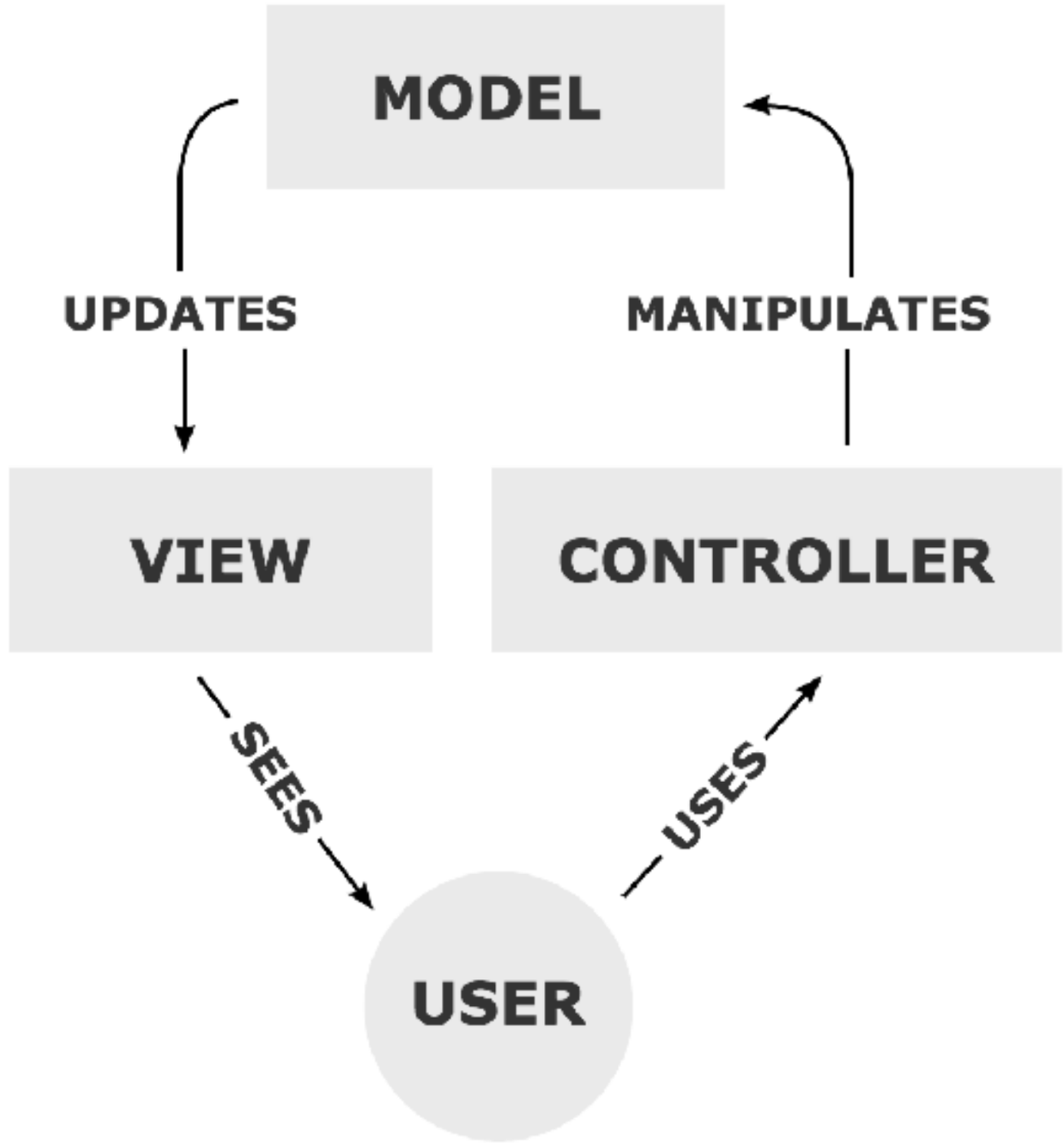
```
switch (answer.result) {
  case Result.DIE:
    onExit();
    die(gameState);
    break;
  case Result.NEXT:
    gameState = setCurrentLevel(gameState, gameState.level + 1);
    update();
    break;
  case Result.WIN:
    onExit();
    end();
    break;
  default:
    throw new Error(`Unknown result: ${answer.result}`);
}
```

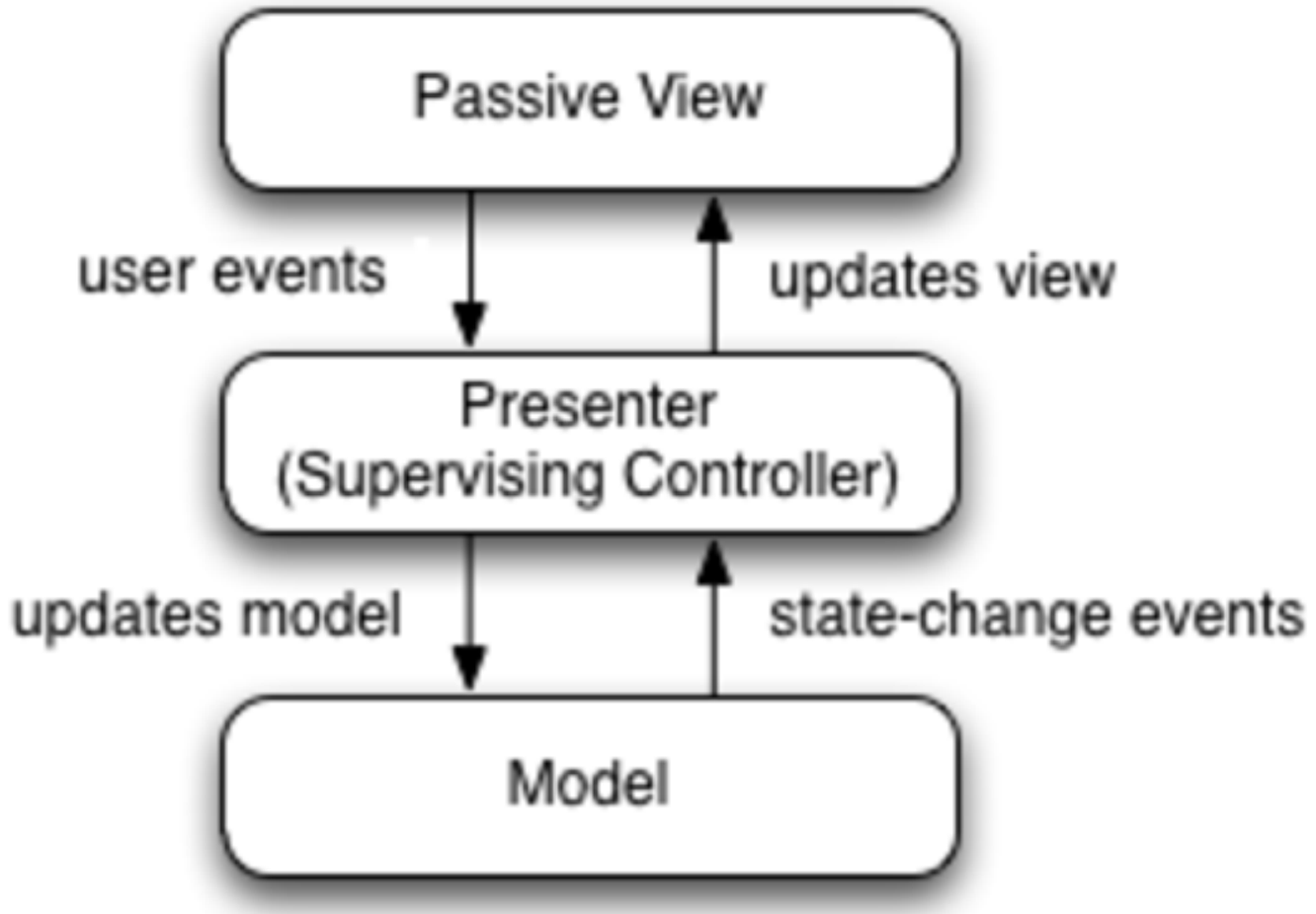
Паттерн Ленивое Вычисление (lazy-loading)

```
export default class AbstractView {

  get element() {
    if (!this._element) {
      this._element = createElement(this.getMarkup());
      this.bindHandlers();
    }
    return this._element;
  }

  getMarkup() {
    throw new Error(`Abstract method should be implemented`);
  }

  bindHandlers() {
    // By default there is nothing to bind
  }

  clearHandlers() {
    // By default nothing to clear
  }
}
```

MVC

Model-View-Controller


MODEL
UPDATES
MANIPULATES
VIEW
CONTROLLER
SEES
USES
USER


Passive View
user events
updates view
Presenter
(Supervising Controller)
updates model
state-change events
Model

UML (Unified Modeling Language)

язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур

ru.wikipedia.org/wiki/UML

UML

- **Диаграмма классов**
- Диаграмма компонентов
- Диаграмма объектов
- Диаграмма деятельности
- Диаграмма автомата
- Диаграмма сценариев использования
- Диаграммы коммуникации и последовательности
- И многие другие

Диаграмма классов

диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними

ru.wikipedia.org/wiki/Диаграмма_классов

Диаграмма классов: Класс

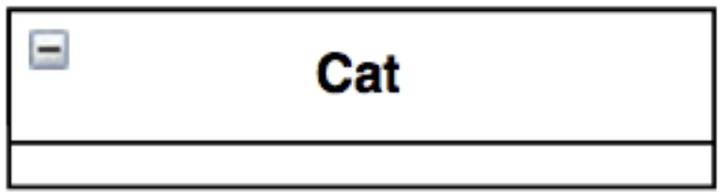
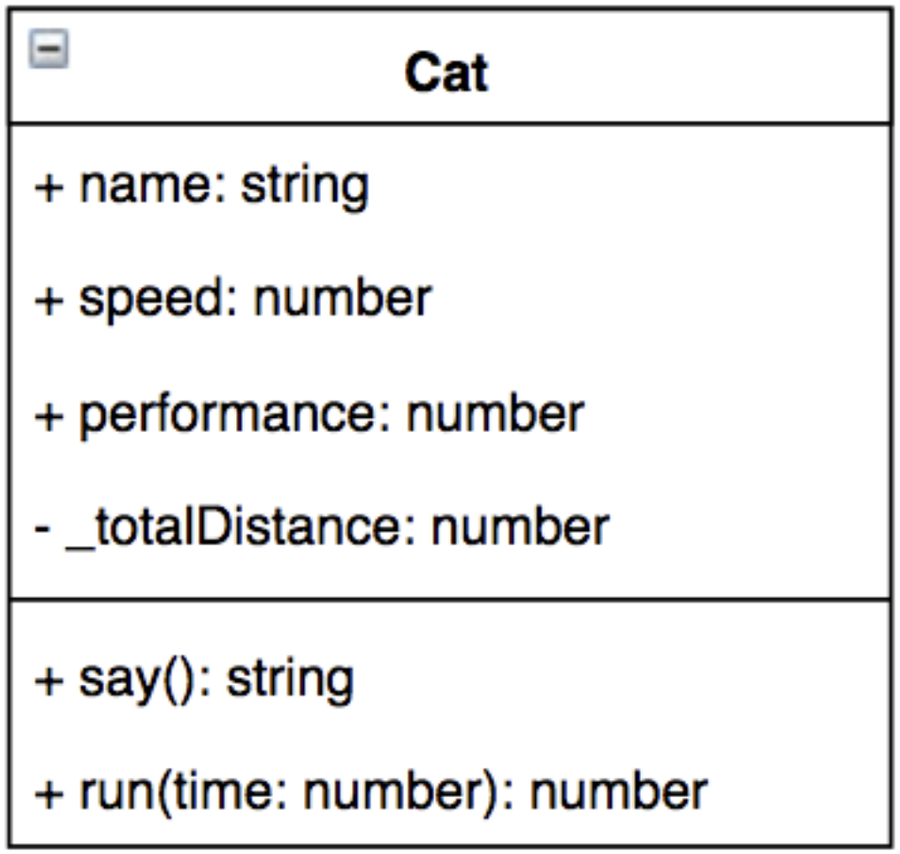


Диаграмма классов: Класс



Реализация 1

```
class Cat {
  constructor(name, speed = 3) {
    this.name = name;
    this.speed = speed;

    this._totalDistance = 0;
  }

  get performance() {
    return this._totalDistance / this.age;
  }

  say() {
    return `${this.name} говорит: 'Мяу'!`;
  }

  run(time) {
    const distance = time * this.speed;
    this._totalDistance += distance;
    return distance;
  }
}
```

Реализация 2

```
class Cat {
  constructor(name, speed = 3) {
    this.name = name;
    this.speed = speed;

    this._totalDistance = 0;
  }

  get performance() {
    return this._totalDistance;
  }

  say() {
    return 'Мяу!';
  }

  run(time) {
    this._totalDistance += time * this.speed;
    return this._totalDistance;
  }
}
```

Диаграмма классов: Связи

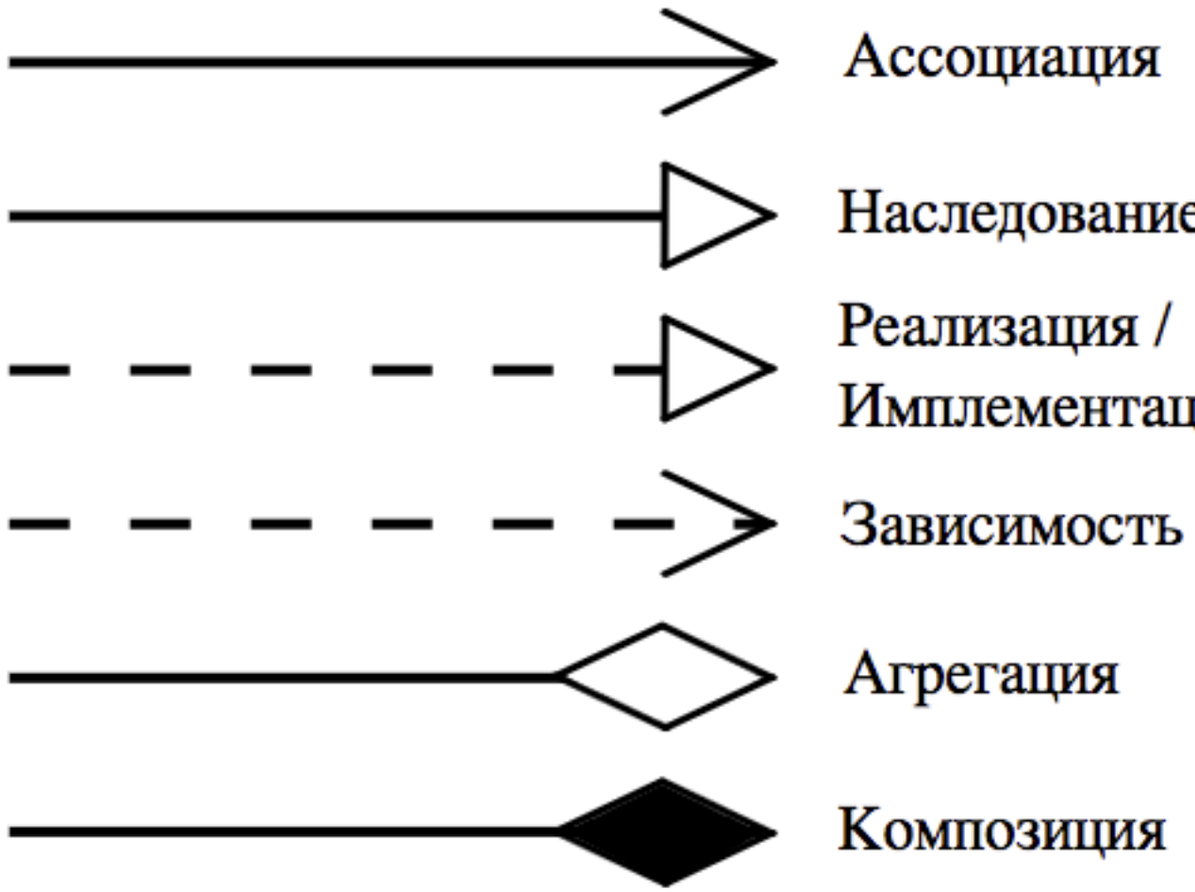
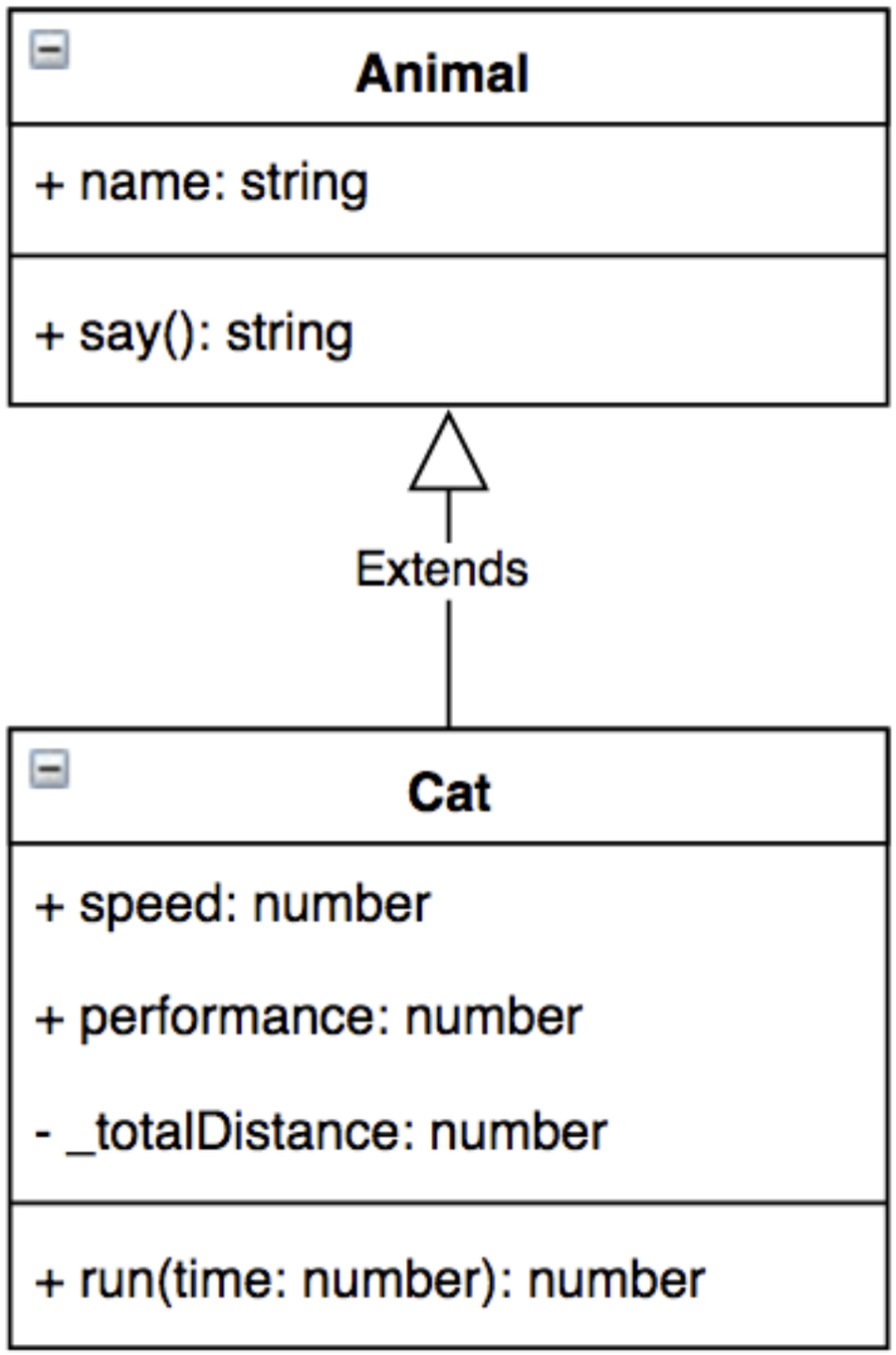


Диаграмма классов: Наследование



Реализация

```
class Cat extends Animal {
  constructor(name, speed = 3) {
    super(name);
    this.speed = speed;

    this._totalDistance = 0;
  }

  get performance() {
    return this._totalDistance / this.age;
  }

  say() {
    return `${this.name} говорит: 'Мяу'!`;
  }

  run(time) {
    const distance = time * this.speed;
    this._totalDistance += distance;
    return distance;
  }
}
```

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  say() {
    throw new Error('Abstract method called');
  }
}
```

Диаграмма классов: Композиция

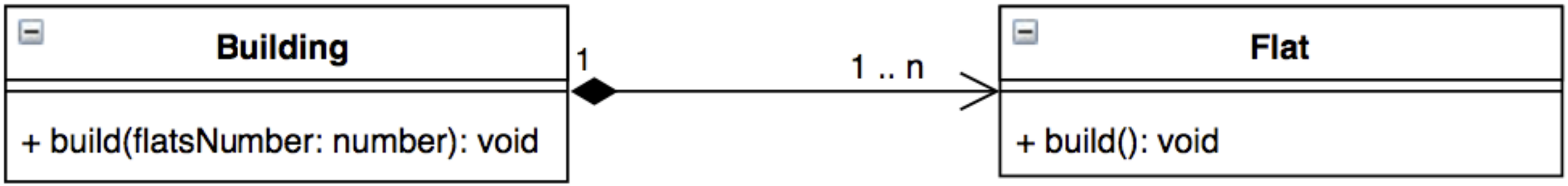


Диаграмма классов: Агрегация

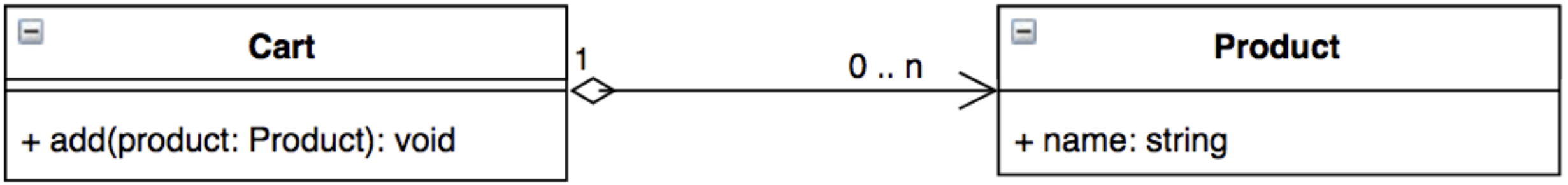
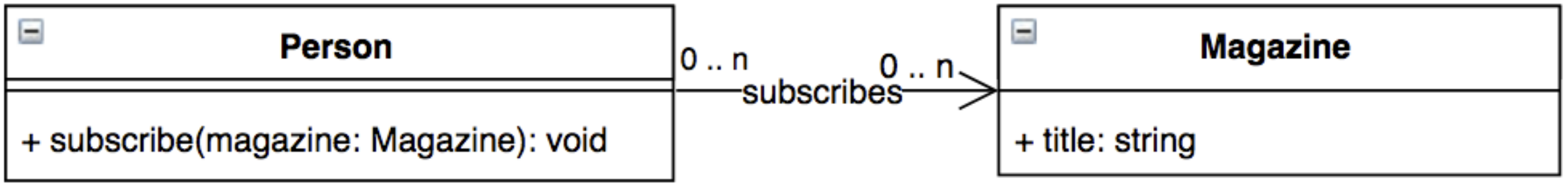
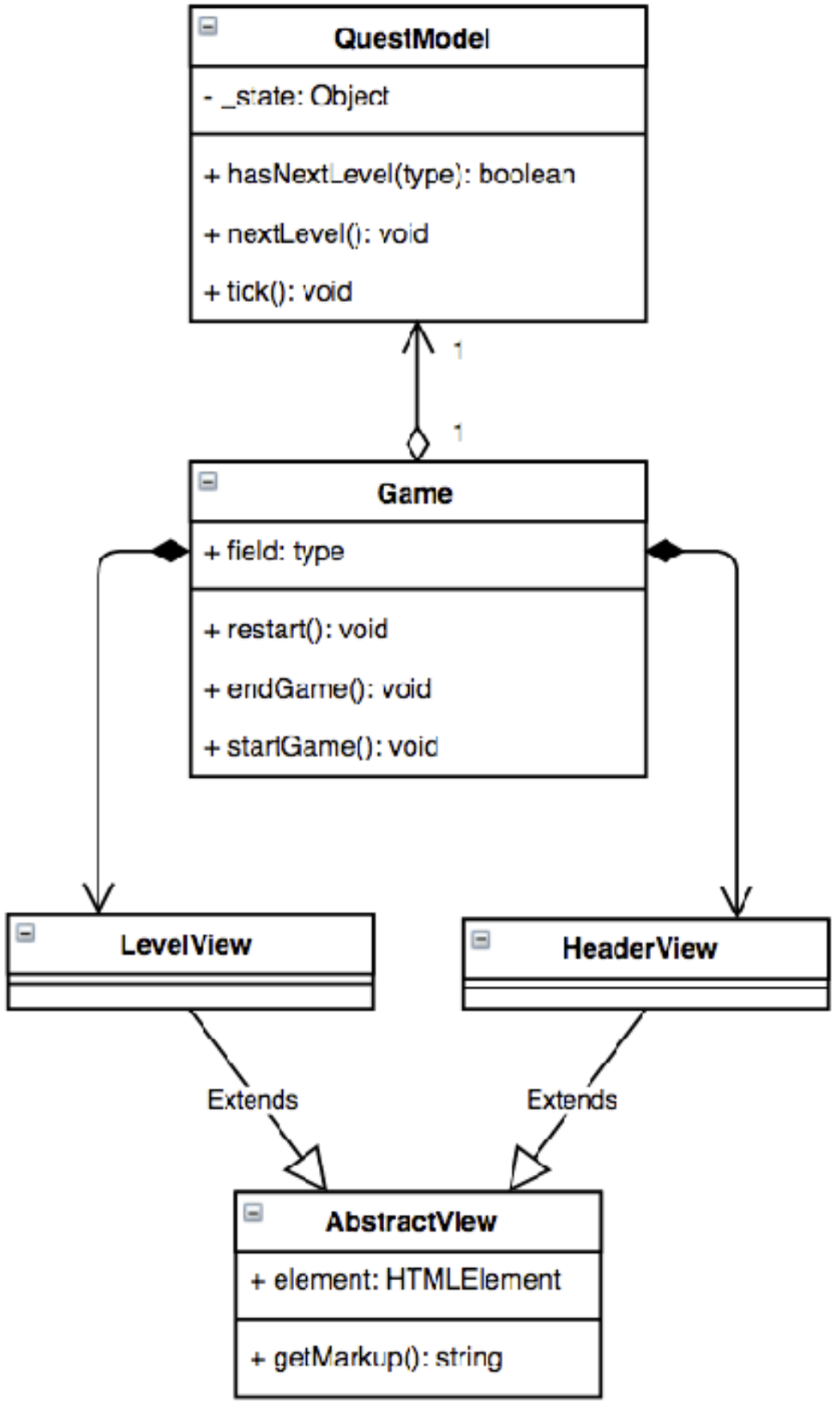


Диаграмма классов: Ассоциация



MVC


QuestModel
- _state: Object
+ hasNextLevel(type): boolean
+ nextLevel(): void
+ tick(): void
1
1
Game
+ field: type
+ restart(): void
+ endGame(): void
+ startGame(): void
LevelView
HeaderView
Extends
Extends
AbstractView
+ element: HTMLElement
+ getMarkup(): string

10 PRINT "До зустрічі!"