Особенности объектно-ориентированных языков программирования

Рахимов И. Р.

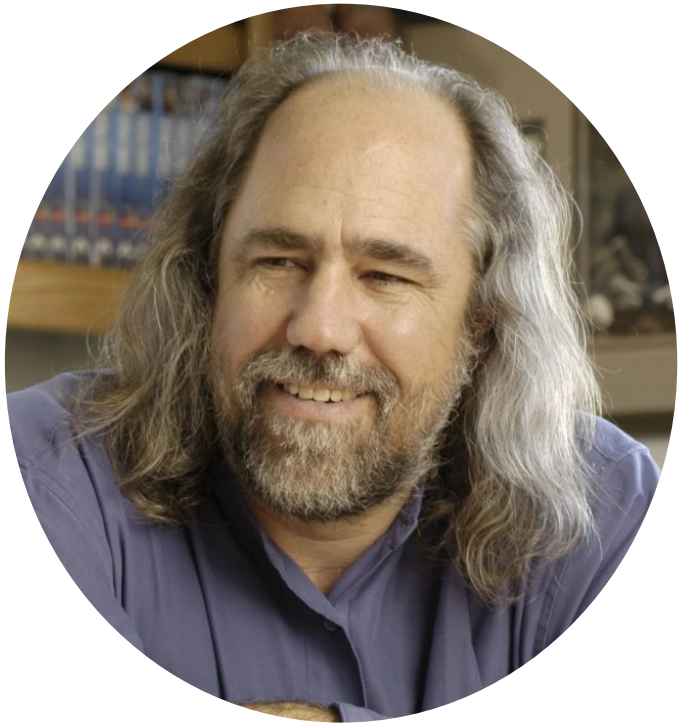
Объекты

две цели:

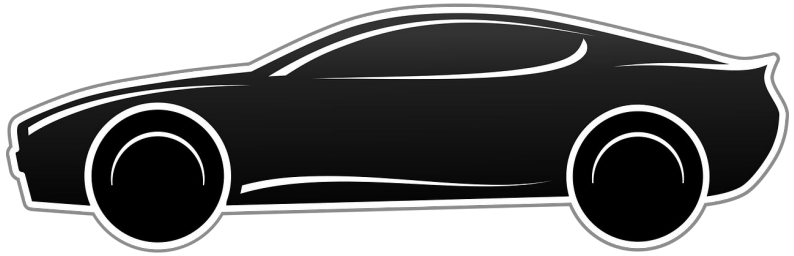
- понимание прикладной задачи (проблемы);
- введение основы для реализации на компьютере.

Объекты

Объект - это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области.



Объект класса



Атрибуты:

- Цвет
- Мощность
- Количество дверей
- Номер автомобиля

Функции (методы):

- Вождение
- Торможение
- Заправка

Состояние (state)

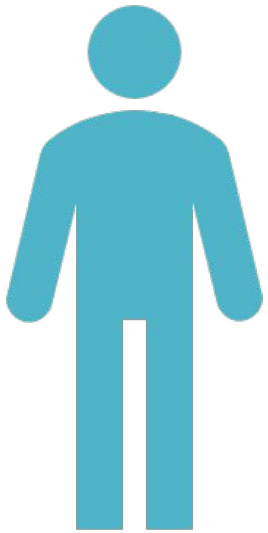
Состояние тесно связано с объектами.

Состояние объекта может определяться наличием или отсутствием связей между моделируемым объектом и другими объектами.

Состояние (state)

Состояние (state) - совокупный результат поведения объекта: одно из стабильных условий, в которых объект может существовать, охарактеризованных количественно;

Состояние. Пример



Объект человек

Объект удочка

Пример в коде

```
void init() async {
  _currentPage = 1;
  var _bothCharacterPage =
      await _characterRepository.getCharacterAsync(_currentPage);
  final characterPage = _bothCharacterPage.data;

  if ((characterPage != null) && (_bothCharacterPage.error == null)) {
    _pagesCount = characterPage.pagesCount;
    _characterLoaded.addAll(characterPage.character);
    emit(CharacterLoadState(character: _characterLoaded));
  } else if (_bothCharacterPage.error != null) {
    emit(NetworkError());
  }
 }
}
```

Поведение

Результат выполнения действий зависит от состояния объекта на момент совершения действия, т.е. нельзя, например, удалить файл, если он открыт кем-либо (заблокирован).

Поведение

Программа, написанная с использованием ООП, обычно состоит из множества объектов, и все эти объекты взаимодействуют между собой.

Уникальность

Identity (уникальность) объекта состоит в том, что всегда можно определить, указывают две ссылки на один и тот же объект или на разные объекты.

В машинном представлении под параметром уникальности объекта чаще всего понимается адрес размещения объекта в памяти.

Классы

Все объекты одного и того же класса описываются одинаковыми наборами атрибутов. Однако объединение объектов в классы определяется не наборами атрибутов, а семантикой.

Классы

Класс — это шаблон поведения объектов определенного типа с заданными параметрами, определяющими состояние. Все экземпляры одного класса (объекты, порожденные от одного класса) имеют один и тот же набор свойств и общее поведение, то есть одинаково реагируют на одинаковые сообщения.

Классы

Класс — это шаблон поведения объектов определенного типа с заданными параметрами, определяющими состояние. Все экземпляры одного класса (объекты, порожденные от одного класса) имеют один и тот же набор свойств и общее поведение, то есть одинаково реагируют на одинаковые сообщения.

Классы. Пример

Барсик

Белла

Борис

Классы. Пример

Пример класса в коде

```dart
class Car {
  String color = "";
  int power = 0;
  int doorsNumber = 0;
  String carNumber = "";

  ride() {
    //код выполняемой функции
    //которая что-возвращает
  }

  void braking() {
    //код выполняемой функции
  }

  void refueling() {
    //код выполняемой функции
  }
}
```

```dart
void main() {
  Car lada = Car();
  lada.carNumber = 'e105df';
  lada.color = 'red';
  lada.doorsNumber = 4;
  lada.power = 400;

  var rideResult = lada.ride();
}
```

Икапсуляция

Инкапсуляция (encapsulation) - это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса).

Икапсуляция

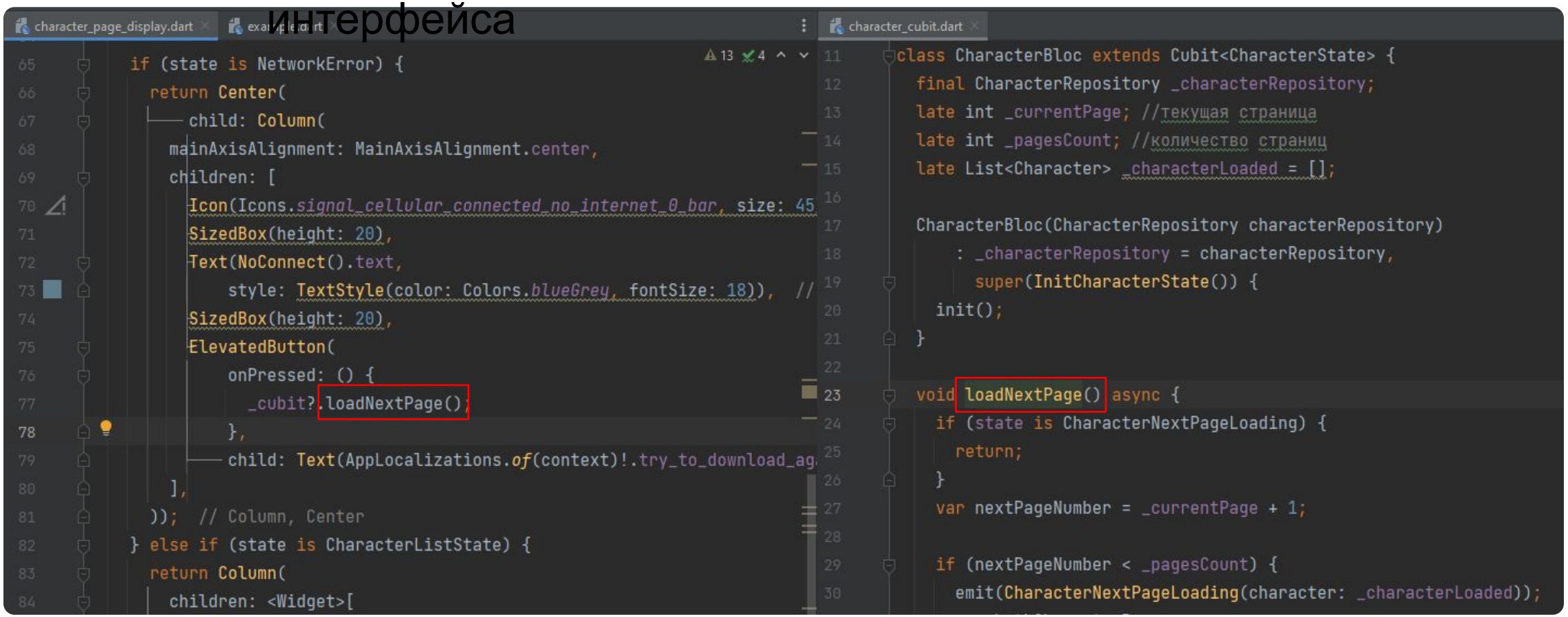
1. При использовании объектно-ориентированного подхода не принято применять прямой доступ к свойствам какого-либо класса из методов других классов.

2. Для доступа к свойствам класса принято задействовать специальные методы этого класса для получения и изменения его свойств.

Пример (инкапсуляция)

Файл интерфейса

Файл бэкенда



character_page_display.dart

```
if (state is NetworkError) {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(Icons.signal_cellular_connected_no_internet_0_bar, size: 45
        SizedBox(height: 20),
        Text(NoConnect().text,
            style: TextStyle(color: Colors.blueGrey, fontSize: 18)),  //
        SizedBox(height: 20),
        ElevatedButton(
            onPressed: () {
              _cubit?.loadNextPage();
            },
            child: Text(AppLocalizations.of(context)!.try_to_download_ag
      ],
    ));  // Column, Center
} else if (state is CharacterListState) {
  return Column(
    children: <Widget>[
```

character_cubit.dart

```
class CharacterBloc extends Cubit<CharacterState> {
  final CharacterRepository _characterRepository;
  late int _currentPage; //текущая страница
  late int _pagesCount; //количество страниц
  late List<Character> _characterLoaded = [];

  CharacterBloc(CharacterRepository characterRepository)
      : _characterRepository = characterRepository,
        super(InitCharacterState()) {
    init();
  }

  void loadNextPage() async {
    if (state is CharacterNextPageLoading) {
      return;
    }
    var nextPageNumber = _currentPage + 1;

    if (nextPageNumber < _pagesCount) {
      emit(CharacterNextPageLoading(character: _characterLoaded));
```

Пример (инкапсуляция)

Файл

Файл бэкенда

character_page_display.dart | example.dart

```dart
if (state is NetworkError) {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(Icons.signal_cellular_connected_no_internet_0_bar, size: 45
        SizedBox(height: 20),
        Text(NoConnect().text,
            style: TextStyle(color: Colors.blueGrey, fontSize: 18)),  //
        SizedBox(height: 20),
        ElevatedButton(
            onPressed: () {
              _cubit?._loadNextPage();
            },
            child: Text(AppLocalizations.of(context)!.try_to_download_ag
      ],
    ));  // Column, Center
} else if (state is CharacterListState) {
  return Column(
    children: <Widget>[
      Expanded(
```

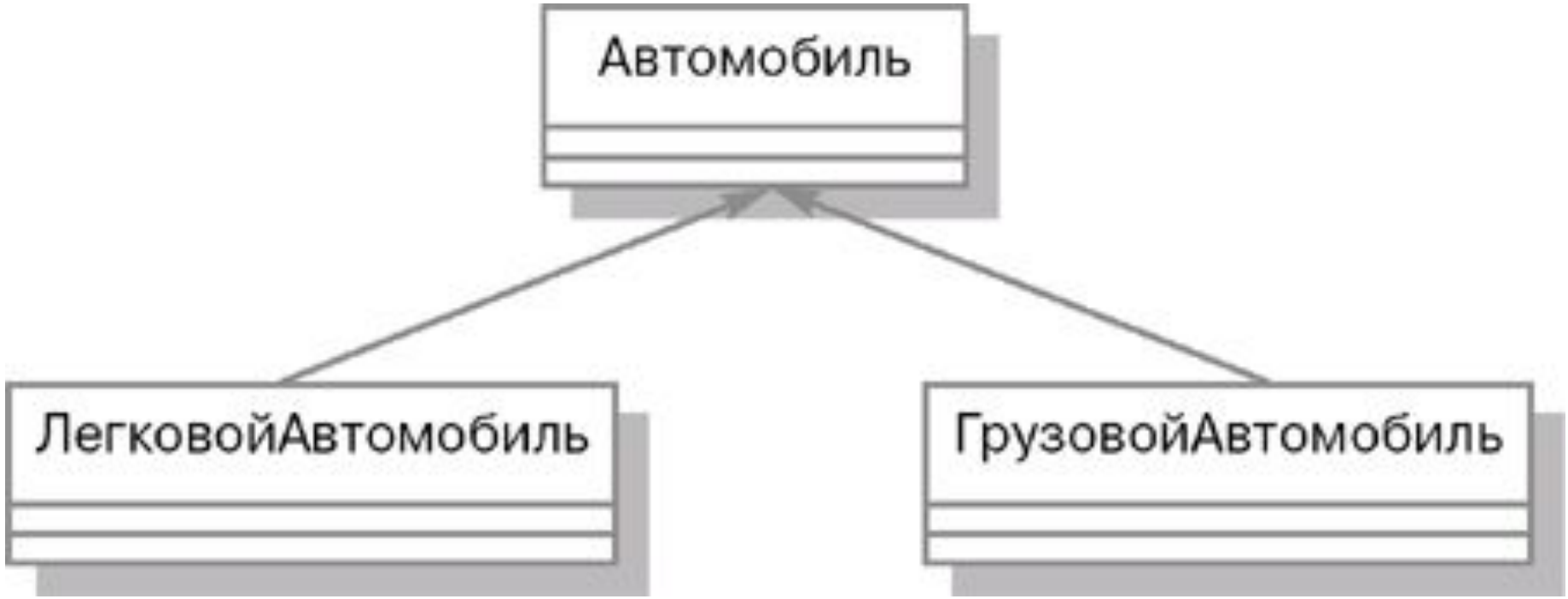
character_cubit.dart

```dart
class CharacterBloc extends Cubit<CharacterState> {
  final CharacterRepository _characterRepository;
  late int _currentPage; //текущая страница
  late int _pagesCount; //количество страниц
  late List<Character> _characterLoaded = [];

  CharacterBloc(CharacterRepository characterRepository)
      : _characterRepository = characterRepository,
        super(InitCharacterState()) {
    init();
  }

  void _loadNextPage() async {
    if (state is CharacterNextPageLoading) {
      return;
    }
    var nextPageNumber = _currentPage + 1;

    if (nextPageNumber < _pagesCount) {
      emit(CharacterNextPageLoading(character: _characterLoaded));
      var bothCharacterPage =
```

Наследование

Наследование (inheritance) - это отношение между классами, при котором класс использует структуру или поведение другого класса (одиночное наследование), или других (множественное наследование) классов.

Наследование



Наследование

Наследование вводит иерархию "общее/частное", в которой подкласс наследует от одного или нескольких более общих суперклассов. Подклассы обычно дополняют или переопределяют унаследованную структуру и поведение.

Наследование. Пример

```dart
class Character {
  final int id;
  final String name;
  final String imageUrl;

  const Character(
      {required this.id,
        required this.name,
        required this.imageUrl});
}
```

```dart
class CharacterDetailed extends Character {
  CharacterDetailed(
      {required super.id,
      required super.name,
      required super.imageUrl,
      required this.status,
      required this.gender,
      required this.locationName});

  final String status;
  final String gender;
  final String locationName;
}
```

Полиморфизм

Полиморфизм (polymorphism) - положение теории типов, согласно которому имена (например, переменных) могут обозначать объекты разных (но имеющих общего родителя) классов. Следовательно, любой объект, обозначаемый полиморфным именем, может по-своему реагировать на некий общий набор операций

Полиморфизм. Пример

```
...
//создание пустого массива, который может
// содержать объекты Point с максимальным
// объемом 1000
Point[] p = new Point[1000];

Line[] l = new Line[1000];
Circle[] c = new Circle[1000];
Box[] b = new Box[1000];
...
// предположим, в этом месте происходит
// заполнение всех массивов соответствующими
// объектами
...
for(int i = 0; i < p.length;i++) {
```

Полиморфизм. Пример

```
...
for(int i = 0; i < p.length;i++) {
//цикл с перебором всех ячеек массива.
   //вызов метода draw() в случае,
   // если ячейка не пустая.
   if(p[i]!=null) p[i].draw();
}

for(int i = 0; i < l.length;i++) {
   if(l[i]!=null) l[i].draw();
}

for(int i = 0; i < c.length;i++) {
   if(c[i]!=null) c[i].draw();
}
```

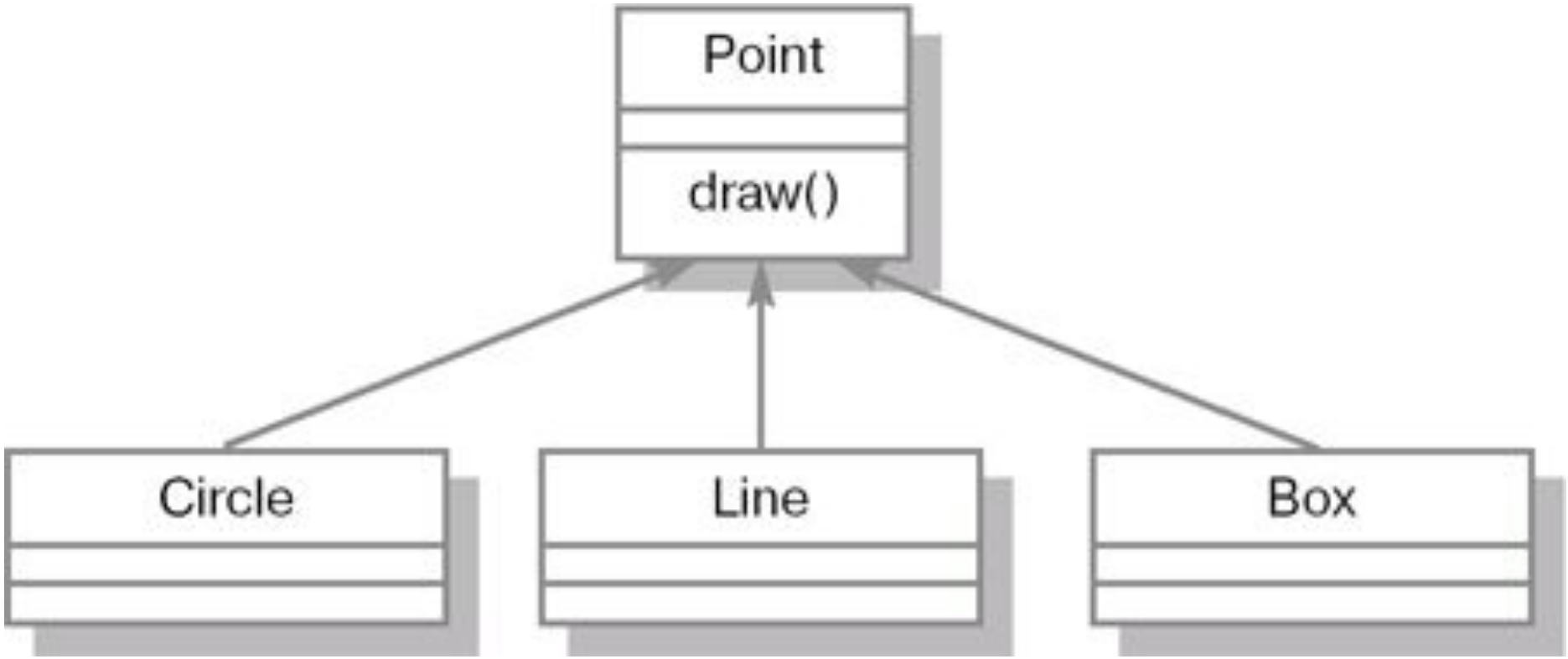
Полиморфизм. Пример

```
for(int i = 0; i < l.length;i++) {
   if(l[i]!=null) l[i].draw();
}

for(int i = 0; i < c.length;i++) {
   if(c[i]!=null) c[i].draw();
}

for(int i = 0; i < b.length;i++) {
   if(b[i]!=null) b[i].draw();
}
...
```

Полиморфизм. Пример

```
...
Point p[] = new Point[1000];
p[0] = new Circle();
p[1] = new Point();
p[2] = new Box();
p[3] = new Line();
...
for(int i = 0; i < p.length;i++) {
   if(p[i]!=null) p[i].draw();
}
...
```

Полиморфизм



Полиморфизм

Процедурный полиморфизм предполагает возможность создания нескольких процедур или функций с одним и тем же именем, **но разным количеством** или **различными типами передаваемых параметров**.

Полиморфизм

Такие одноименные функции называются перегруженными, а само явление - перегрузкой (overloading).

Перегрузка функций существует и в ООП и называется перегрузкой методов.

Достоинства ООП

- Классы позволяют проводить конструирование из полезных компонентов, обладающих простыми инструментами, что позволяет абстрагироваться от деталей реализации.

Достоинства ООП

- Данные и операции над ними образуют определенную сущность, и они не разносятся по всей программе, как нередко бывает в случае процедурного программирования, а описываются вместе. Локализация кода и данных улучшает наглядность и удобство сопровождения программного обеспечения.

Достоинства ООП

- Инкапсуляция позволяет привнести свойство модульности, что облегчает распараллеливание выполнения задачи между несколькими исполнителями и обновление версий отдельных компонентов.

Недостатки ООП

- Сложность в освоении. ООП сложнее, чем функциональное программирование. Для написания кода в этой парадигме нужно знать гораздо больше. Поэтому перед созданием первой рабочей программы придётся освоить много информации: разобраться в классах и наследовании, научиться писать публичные и внутренние функции, изучить способы взаимодействия объектов между собой.

Недостатки ООП

- Громоздкость. Там, где в функциональном программировании хватит одной функции, в ООП нужно создать класс, объект, методы и атрибуты. Для больших программ это плюс, так как структура будет понятной, а для маленьких может оказаться лишней тратой времени.

Недостатки ООП

- Низкая производительность (не всегда). Объекты потребляют больше памяти, чем простые функции и переменные. Скорость компиляции от этого тоже страдает.