Объектно-ориентированное программирование

1. Понятие ООП
2. Принципы объектно-ориентированного программирования
3. Конструкторы
4. Ключевое слово this
5. Различия между процедурным и объектно-ориентированным подходами

1. Понятие ООП

- **Объектно-ориентированное программирование (ООП) – подход к созданию программ,** основанный на использовании классов и объектов, взаимодействующих между собой.
- **Объект** — это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области.

Гради Буч

- Примеры объектов:
  - *Осязаемые объекты*: стол, бильярдный шар, компьютер;
  - *Неосязаемые объекты, события или явления*: химический процесс, траектория движения шара.
- Объект обладает состоянием и поведением.

- Состояние объекта характеризуется перечнем всех свойств данного объекта и текущими значениями каждого из этих свойств.
- Свойство — это характеристики, черты, качества или способности, делающие данный объект самим собой.
- Пример состояния некоторой окружности

Свойство	Значение
Центр	(0,2)
Радиус	5
цвет	«красный»

- Состояние объекта может измениться только в результате вызова методов.
- Поведение – это то, как объект действует или реагирует.
- Поведение – это набор операций (методов) объекта.
- Операция – это услуга, которую объект может предоставить своим клиентам (другим объектам).

Классы

- Класс – это некоторое множество объектов, имеющих общую структуру и общее поведение. Любой объект является экземпляром класса.
- Все экземпляры одного класса будут вести себя одинаковым образом в ответ на одинаковые запросы.
- Класс — множество объектов с общей структурой и поведением.

Определение класса

- Класс определяется с помощью ключевого слова class:

```
class Book{

}
```

- Состояние объекта определяется свойствами класса.
- Поведение объекта определяется методами класса.

```
class Circle {

public double x;
public double y;
public double r;

public void printCircle(){
System.out.println("Окружность с центром
   ("+x+";"+y+") и радиусом "+r);

 }
}
```

Свойства класса

Метод класса

Создание программы в рамках ООП:

1. Из предметной области задачи выделяются существенные свойства, *с учётом* которых создаются классы.
2. От классов порождаются объекты, обладающие некоторым начальным состоянием (значениями свойств).
3. Объекты начинают взаимодействовать между собой с помощью методов, изменяя своё состояние.
4. Так, получается модель некоторого явления или процесса. Чтобы получить полезный результат, надо оценить состояние этой модели в нужный момент.

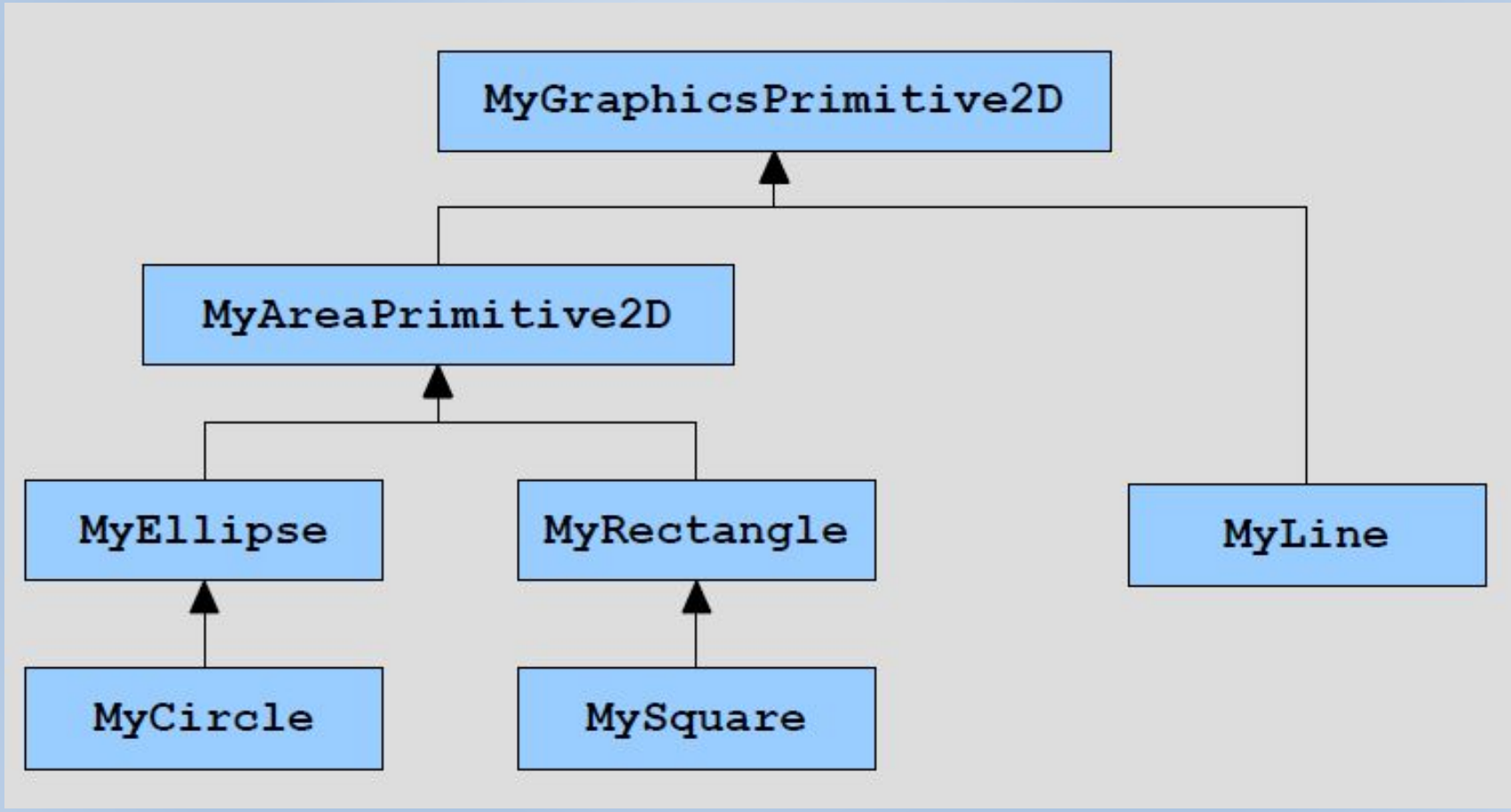
2.Принципы объектно-ориентированного подхода

- Класс должен проектироваться и разрабатываться с использованием принципов инкапсуляции, наследования и полиморфизма.
- Правильное применение указанных принципов повышает абстрагирование и улучшает классификацию предметной области (программы), что в конечном счете позволят бороться со сложностью предметной области.

Понятие абстрагирования и классификации

- Абстрагирование (abstraction) — это представление предметной области (программы) в виде меньшего количества более крупных понятий (блоков) и минимизация связей между ними.
- Классификация — это выделение существенных, устойчивых отношений (связей) между понятиями (блоками).

Примеры абстрагирования и классификации



Понятие инкапсуляции

- Инкапсуляция - это объединение данных с функциями их обработки в сочетании с сокрытием ненужной для использования этих данных информацией.
- Инкапсуляция - это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса).

Понятие интерфейса и реализации

- Интерфейс (interface) – это внешний вид класса, выделяющий его существенные черты и не показывающий внутреннего устройства и секретов поведения.
- Реализация (implementation) – внутреннее представление класса, включая секреты его поведения.

Преимущества использования инкапсуляции

- Повышает степень абстракции программы — для написания программы не требуется знания данных класса и реализация его функций.
- Позволяет изменить реализацию класса без модификации остальной части программы, если интерфейс остался прежним.

Модификаторы доступа

- Для решения проблемы сокрытия реализации в Java используются модификаторы доступа. Уровни доступа варьируются от «доступа ко всему» до минимального.

- В Java существуют следующие модификаторы доступа:
  - ✔ public
  - ✔ protected
  - ✔ private
  - ✔ доступ в пределах пакета

Модификатор public

- Модификатор ***public*** определяет, что следующее за ним объявление **доступно всем**, и в том числе произвольным программам клиентам.

```
public class Cookie {                                   // Класс Печенье
   public void getApple() { ... }
   private void getOrange() { ... }
}

public class Dinner {
   public void run() {
      cookie.getApple();                                // метод доступен
      cookie.getOrange();                               // метод недоступен
   }
}
```

Модификатор private

- Модификатор ***private*** обозначает, что **никто не имеет право** получить доступ к этому члену за исключением его класса, изнутри методов этого класса.

```
public class Cookie {
   private void getOrange() { ... }

   public void makeJuice() {
      getOrange();                                  // метод доступен
   }
}

public class Dinner {
   public void run() {
      cookie.getOrange();                           // метод недоступен
   }
}
```

Модификатор protected

- Модификатор ***protected*** обозначает, что **никто не имеет право** получить доступ к этому члену **за исключением его** класса, и классов **унаследованных** от него.

```
public class Cookie {
    protected void getOrange() { ... }
}

public class Cake extends Cookie {
    public void run() {
        getOrange();              // метод доступен
    }
}

public class Dinner {
    public void run() {
        cookie.getOrange();     // метод  недоступен
    }
}
```

Модификатор «в пределах пакета»

- Если вообще не указывать модификатор, то мы получим доступ по умолчанию – **в пределах пакета**. Поле будет доступно только классам в этом пакете.

- **package ru.sbs.jc;**

```
public class Cookie {
    void getOrange() { ... }
}
```

- **package ru.sbs.jc;**

```
public class Cake {
    public void run() {
        cookie.getOrange();        // метод доступен
    }
}
```

Понятие наследования

- Наследование – это такое отношение между классами, когда один класс повторяет структуру и поведение другого класса . Другими словами, структура и поведение передается от предка к потомку.
- Наследование реализует отношение "is-a" между двумя классами, т.е. дочерний класс должен быть частным или специализированным случаем родительского класса.

- Наследование в Java позволяет повторно использовать код одного класса в другом классе, т.е. можно унаследовать новый класс от уже существующего класса. Главный наследуемый класс в Java называют родительским классом, или суперклассом. Наследующий класс называют дочерним классом, или подклассом. Подкласс наследует все поля и свойства суперкласса, а также может иметь свои поля и свойства, отсутствующие в классе-родителе.
- Создание подкласса выполняется с помощью ключевого слова extends.

```
class Siemens
{
int w, h;
int Area()
{
return w*h;
}
}
```

// родительский класс (суперкласс)

```
class SiemensM55 extends Siemens
{
// члены класса
}
```

// подкласс

Преимущества использования наследования

- При наследовании общие свойства и поведение не описываются, что сокращает объем программы.
- Выделение общих черт различных классов в один класс-предок является мощным механизмом абстракции и классификации.

Конструктор подкласса

- Автоматически вызывается конструктор суперкласса без аргументов. Если такой конструктор у суперкласса отсутствует, возникает ошибка.
- Для вызова конструктора суперкласса – первой командой в коде конструктора подкласса должно быть ключевое слово super().

```
public class Student extends User {
    int group;

public Student (int age, String firstName, String
lastName, int group) {
    super(age, firstName, lastName);
    this.group=group;
    }
```

Закрытые члены класса

- Члены (поля, методы), объявленные в суперклассе с ключевым словом private, в подклассе не наследуются.

Понятие полиморфизма

- Полиморфизм — это использование одного имени для различных сущностей. При этом разнородные сущности, выступая под одним именем, воспринимаются как однотипные.
- Полиморфизм — это возможностью обработки данных переменного типа.
- Полиморфизм — это возможность оперировать объектами, не обладая точным знанием их типов.

Преимущества использования полиморфизма

- Позволяет записывать алгоритмы лишь однажды и затем повторно их использовать для различных типов данных, которые, возможно, еще не существуют (обобщенные алгоритмы или обобщенное программирование).
- Сужает концептуальное пространство, т.е. уменьшает количество информации, которое необходимо помнить программисту.

Перегрузка методов

- Является разновидностью полиморфизма.
- Перегрузка методов – использование методов с одинаковыми именами, но разными аргументами.

Каждый перегруженный метод должен иметь уникальный список аргументов.

Мотив – одинаковое поведение для разных типов.

```
public class App {
    public void print(int i) {
        …
    }

    public void print(String st) {
        …
    }
}
```

3. Конструкторы

- В языке Java разработчик класса может в обязательном порядке выполнять инициализацию каждого объекта, используя специальный метод, называемый *конструктором*.

- Имя конструктора совпадает с именем класса, у конструктора могут быть аргументы но нет возвращаемого значения.

```
class Rock {
    int id;

    public Rock (int value) {
        id = value;
    }
}
```

Конструктор по умолчанию

- Когда создается класс без конструктора, компилятор автоматически добавляет конструктор по умолчанию.

```
class Bird {
    int i;
}

public class App {
    public static void main(String[] args) {
        Bird bird = new Bird(); // конструктор по умолчанию
    }
}
```

4. **Ключевое слово this**

- Ключевое слово *this* употребляется только внутри метода и дает ссылку на объект, для которого этот метод был вызван.

- Идентификатор this подразумевает – *этот* объект.

```
public class Flower {
    private int id;

    Flower(int id) {
        this.id = id;
    }

    Flower() {
        this(0);
    }
}
```

5. Различия между процедурным и объектно-ориентированным подходами

- В процедурном подходе с помощью пошагового уточнения исходная задача разбивается на все более мелкие подзадачи, пока они не станут настолько простыми, что их можно будет реализовать непосредственно.
- В объектно-ориентированном подходе сначала выделяются классы, а лишь затем определяются их методы. При этом каждый метод связан с классом и класс отвечает за их выполнение.

- В процедурном подходе программа представляет собой однородное множество процедур.
- В объектно-ориентированном подходе классы предоставляют удобный механизм кластеризации методов. Кроме того скрывают детали представления данных от любого кода, кроме своих методов.
- Это означает, что если ошибка программирования искажает данные, то ее легче найти

- В процедурном подходе невозможно получить несколько копий одного модуля. Модуль - это набор связанных данных и процедур, собранных в отдельном файле. Модуль может иметь интерфейсную часть и реализацию.
- В объектно-ориентированном подходе на основе класса можно создать несколько объектов с одинаковым поведением.

Преимущества объектно-ориентированного подхода

- Более эффективная борьба со сложностью программного обеспечения.
- Более высокий процент повторного использования кода.
- Повышение надежности программного обеспечения.
- Обеспечение возможности модификации отдельных компонентов программного обеспечения без изменения остальных его компонентов.

- Использование объектного подхода существенно повышает уровень унификации разработки и пригодность для повторного использования не только программ, но и проектов, что в конце концов ведет к созданию среды разработки.
- Объектно-ориентированные системы часто получаются более компактными, чем их не объектно-ориентированные эквиваленты. А это означает не только уменьшение объема кода программ, но и удешевление проекта за счет использования предыдущих разработок, что дает выигрыш в стоимости и времени.

- Использование объектной модели приводит к построению систем на основе стабильных промежуточных описаний, что упрощает процесс внесения изменений.
- Это дает возможность развиваться постепенно и не приводит к полной ее переработке даже в случае существенных изменений исходных требований.

- Объектная модель уменьшает риск разработки сложных систем, прежде всего потому, что процесс интеграции растягивается на все время разработки, а не превращается в единовременное событие.
- Объектный подход состоит из ряда хорошо продуманных этапов проектирования, что также уменьшает степень риска и повышает уверенность в правильности принимаемых решений

- Объектная модель ориентирована на человеческое восприятие мира: многие люди, не имеющие понятия о том, как работает компьютер, находят вполне естественным объектно-ориентированный подход к системам.