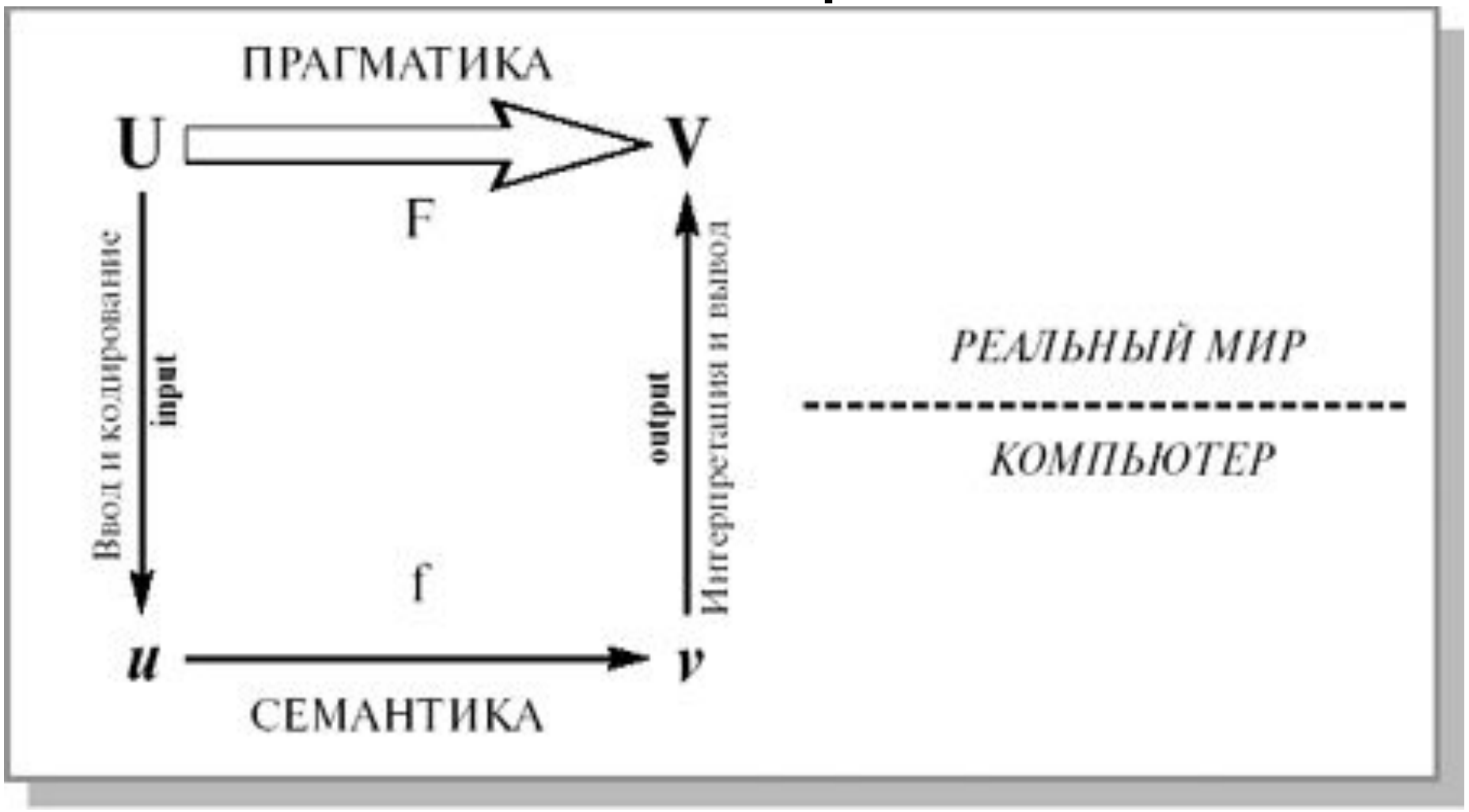
Основы объектно-ориентированного программирования (ООП)

Объектно-ориентированный подход

- ООП основан на представлении **предметной области** задачи в виде множества **моделей** для независимой от языка разработки программной системы на основе ее прагматики.
- **Прагматика** определяется целью разработки программной системы, например, обслуживание клиентов банка, управление работой аэропорта и т.п.
- В формулировке цели участвуют предметы и понятия реального мира, имеющие отношение к создаваемой системе. При объектно-ориентированном подходе эти предметы и понятия заменяются моделями, т.е. определенными формальными конструкциями.

Семантика и прагматика



Определения

- **Семантика** — смысл программы с точки зрения выполняющего ее компьютера.
- **Прагматика** — смысл программы с точки зрения ее пользователей.
- Модель содержит не все признаки и свойства представляемого ею предмета или понятия, а только те, которые существенны для разрабатываемой программной системы.
- Таким образом, модель «беднее», а следовательно, проще представляемого ею предмета или понятия.

Объекты

- **Объект** — понятие, абстракция или любой предмет с четко очерченными границами, имеющий смысл в контексте рассматриваемой прикладной проблемы.
- Введение объектов преследует две цели:
  - понимание прикладной задачи (проблемы);
  - введение основы для реализации на компьютере.
- Примеры объектов: стул, велосипед, Halyk банк.

Объекты

- Каждый объект имеет определенное **время жизни**.
- В процессе выполнения программы, или функционирования какой-либо реальной системы, могут **создаваться** новые объекты и **уничтожаться** уже существующие.
- **Объект** (по Гради Бучу) — это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области.
- Каждый объект имеет **состояние**, обладает четко определенным **поведением** и уникальной **идентичностью**.

Состояние

- Пример: студент сидит, затем прыгает и в то же время выполняет другие действия.
- Состояние объекта может определяться наличием или отсутствием связей между моделируемым объектом и другими объектами.
- Пример: студент и велосипед.
- Для рассмотренных примеров атрибутами объекта «Студент» являются:
  - текущее положение человека (сидит, прыгает);
  - наличие велосипеда (есть или нет).

Состояние

- **Состояние** (state) — совокупный результат поведения объекта: одно из стабильных условий, в которых объект может существовать, охарактеризованных количественно; в любой момент времени состояние объекта включает в себя перечень (обычно статический) свойств объекта и текущие значения (обычно динамические) этих свойств.

Поведение

- Для каждого объекта существует определенный набор действий, которые с ним можно произвести.
- Пример: операции с файлом.
- **Результат** выполнения действий **зависит** от **состояния** объекта на момент совершения действия, т.е. нельзя, например, удалить файл, если он открыт кем-либо (заблокирован).
- В то же время **действия** могут **менять** внутреннее **состояние** объекта — при открытии или закрытии файла свойство «открыт» принимает значения «да» или «нет», соответственно.

Поведение

- Программа, написанная с использованием ООП, обычно состоит из множества объектов, и все эти **объекты взаимодействуют** между собой.
- В терминологии объектно-ориентированного подхода понятия «действие», «сообщение» и «метод» являются синонимами.
- **Поведение** (behavior) — действия и реакции объекта, выраженные в терминах передачи сообщений и изменения состояния; видимая извне и воспроизводимая активность объекта.

Уникальность

- Уникальность — это то, что отличает объект от других объектов.
- Пример: банкноты.
- В машинном представлении под параметром уникальности объекта чаще всего понимается **адрес** размещения объекта в **памяти**.
- Уникальность объекта состоит в том, что всегда можно определить, указывают две ссылки на один и тот же объект или на разные объекты.

Уникальность

- Уникальность ≠ имени ссылки на объект.
- На один объект может указывать несколько ссылок, и ссылки могут менять свои значения (ссылаться на другие объекты).
- **Уникальность** (identity) — свойство объекта; то, что отличает его от других объектов.

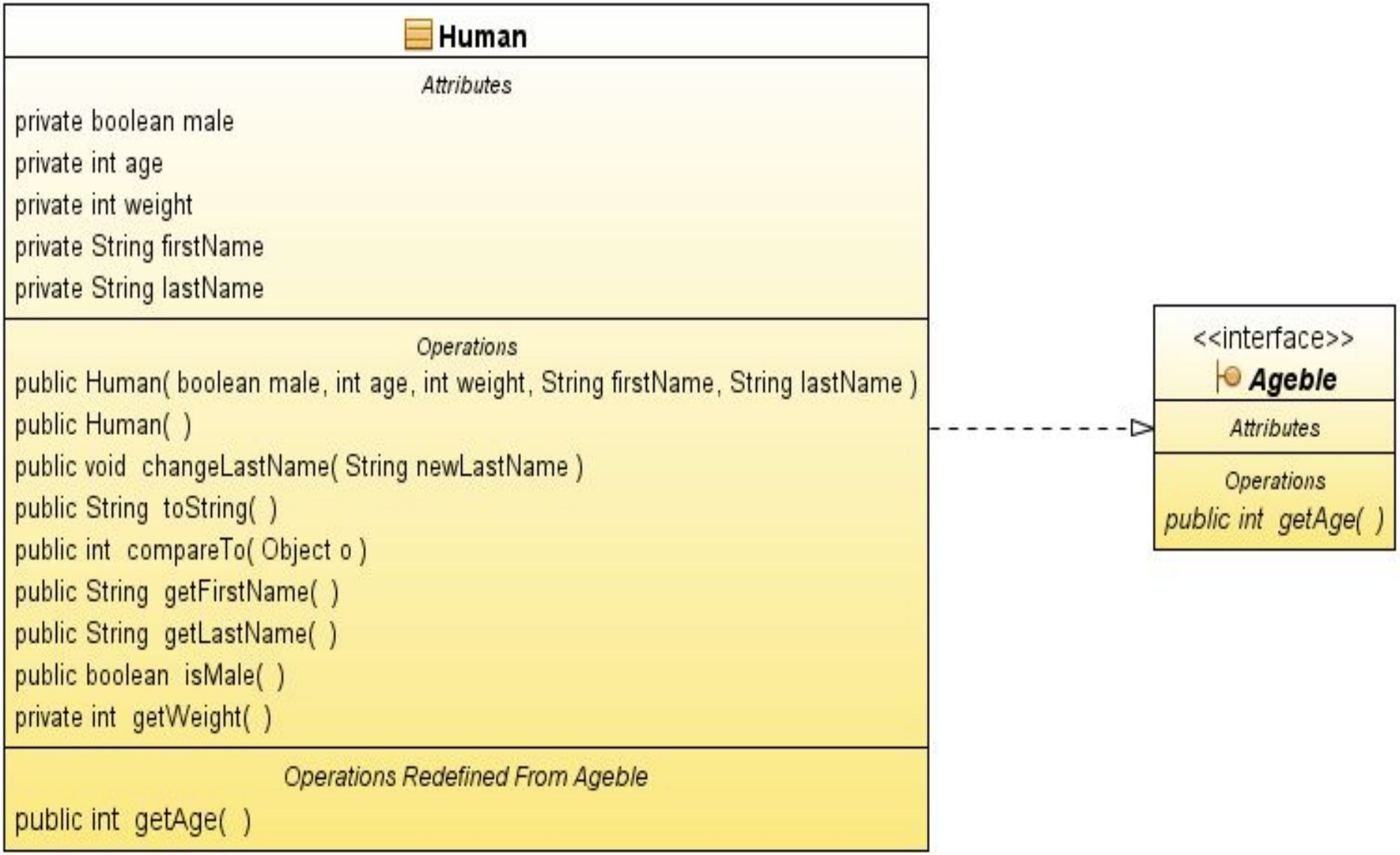
Классы

- Все банкноты принадлежат одному и тому же **классу объектов** (именно с этим связана их одинаковость). Номинальная стоимость, материал, форма — это атрибуты класса.
- Совокупность атрибутов и их значений характеризует объект. Наряду с термином «**атрибут**» часто используют термины «**свойство**», «**реквизит**» и «**поле**».
- Все объекты одного и того же класса описываются одинаковыми наборами атрибутов. Однако объединение объектов в классы определяется не наборами, а семантикой.
- «Конюшня» и «лошадь» могут иметь одинаковые атрибуты: цена и возраст.

Классы

- Формально **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.
- В соответствии с **UML** (Unified Modelling Language — унифицированный язык моделирования), класс изображается в виде прямоугольника, состоящего из трех частей. В верхней части помещается название класса, в средней — свойства объектов класса, в нижней — действия, которые можно выполнять с объектами данного класса (методы).
- Каждый класс также может иметь специальные методы, которые автоматически вызываются при создании и уничтожении объектов этого класса:
  - конструктор (constructor) - выполняется при создании объектов;
  - деструктор (destructor) - выполняется при уничтожении объектов.

Класс Хуман


Human
Attributes
private boolean male
private int age
private int weight
private String firstName
private String lastName
Operations
public Human(boolean male, int age, int weight, String firstName, String lastName)
public Human( )
public void changeLastName(String newLastName)
public String toString( )
public int compareTo(Object o)
public String getFirstName( )
public String getLastName( )
public boolean isMale( )
private int getWeight( )
Operations Redefined From Ageble
public int getAge( )
<<interface>>
Ageble
Attributes
Operations
public int getAge( )

Инкапсуляция

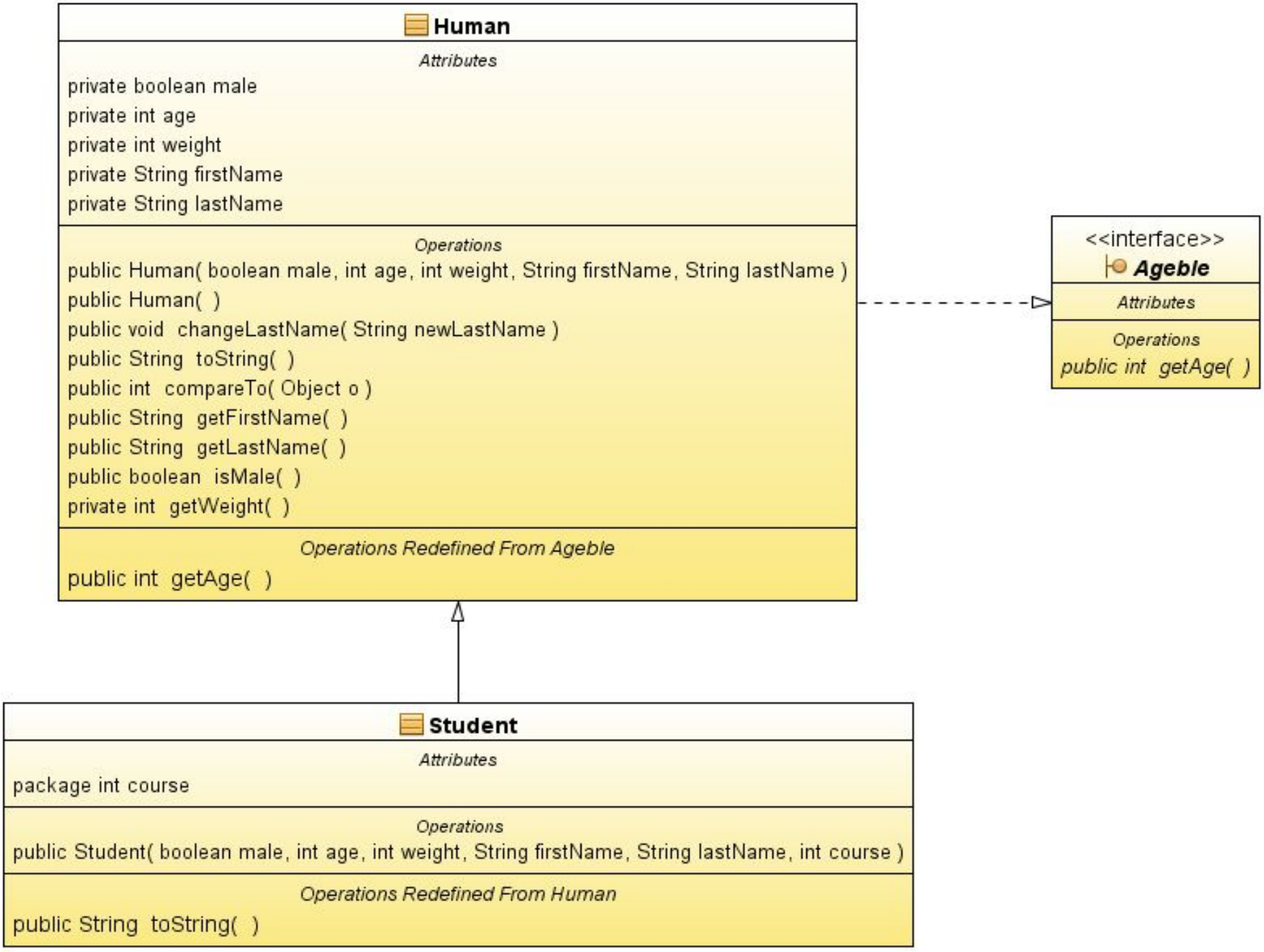
- **Инкапсуляция** (encapsulation) — это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса).
- При использовании объектно-ориентированного подхода не принято применять прямой доступ к свойствам какого-либо класса из методов других классов. Для доступа к свойствам класса принято задействовать специальные методы этого класса для получения и изменения его свойств (getters и setters).
- Внутри объекта данные и методы могут обладать различной степенью открытости (или доступности).

Инкапсуляция

- **Открытые** члены класса составляют **внешний интерфейс** объекта. Это то, что доступно другим классам.
- Благодаря **сокрытию реализации** за внешним интерфейсом класса можно менять внутреннюю логику отдельного класса, не меняя код остальных компонентов системы. Это свойство называется **модульность**.
- Преимущества доступа через методы:
  - контроль корректных значений полей;
  - лёгкость изменения способа хранения данных;
  - простота отладки.

Наследование

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


Human
Attributes
private boolean male
private int age
private int weight
private String firstName
private String lastName
Operations
public Human(boolean male, int age, int weight, String firstName, String lastName)
public Human( )
public void changeLastName(String newLastName)
public String toString( )
public int compareTo(Object o)
public String getFirstName( )
public String getLastName( )
public boolean isMale( )
private int getWeight( )
Operations Redefined From Ageble
public int getAge( )
<<interface>>
Ageble
Attributes
Operations
public int getAge( )
Student
Attributes
package int course
Operations
public Student(boolean male, int age, int weight, String firstName, String lastName, int course)
Operations Redefined From Human
public String toString( )

Наследование

- Отношение обобщения обозначается сплошной линией с треугольной стрелкой на конце. Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее отсутствие - на более специальный класс (класс-потомок или подкласс).
- Использование наследования способствует уменьшению количества кода, а также способствует написанию более эффективного и гибкого кода.
- В рассмотренном примере применено одиночное наследование. Некоторый класс также может наследовать свойства и поведение сразу нескольких классов.

Полиморфизм

- Полиморфизм является одним из фундаментальных понятий в объектно-ориентированном программировании наряду с наследованием и инкапсуляцией.
- Слово «полиморфизм» греческого происхождения и означает «имеющий много форм».

Пример

```
void main() {
    Student[] students = new Student[2];
    Human[] humans = new Human[3];
    for (Student student : students) {
        System.out.println(student.toString());
    }
    for (Human human : humans) {
        System.out.println(human.toString());
    }
}
```

```
Human[] polyHumans = new Human[5];
polyHumans[0] = new Human();
polyHumans[1] = new Student();
for (Human human : polyHumans) {
    System.out.println(human.toString());
}
```

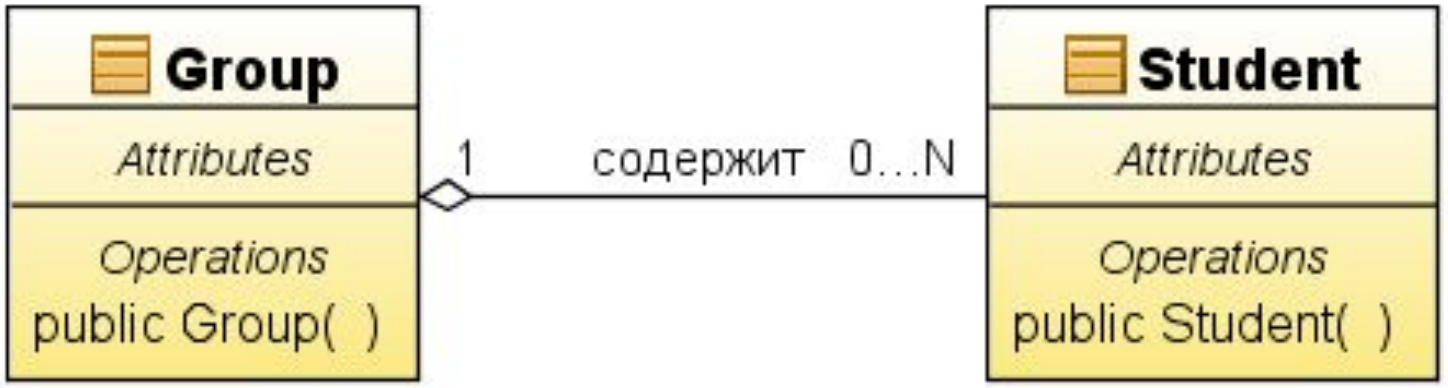
Полиморфизм

- 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.
- **Перегрузка методов** — возможность создания нескольких методов с одним и тем же именем, но разным количеством или различными типами передаваемых параметров.

Типы отношений между классами

- Любая программа, написанная на объектно-ориентированном языке, представляет собой некоторый набор связанных между собой классов.
- Пример: обучение студента.
- Самые распространённые связи между классами в рамках объектной модели:
  - агрегация (Aggregation);
  - ассоциация (Association);
  - наследование (Inheritance);
  - метаклассы (Metaclass).

Агрегация



- Агрегация изображается линией с ромбиком на стороне того класса, который выступает в качестве владельца, или контейнера. Необязательное название отношения записывается посередине линии.
- Объект класса Group содержит несколько объектов Student. В то же время каждый студент «знает», в какой именно он группе.

Агрегация

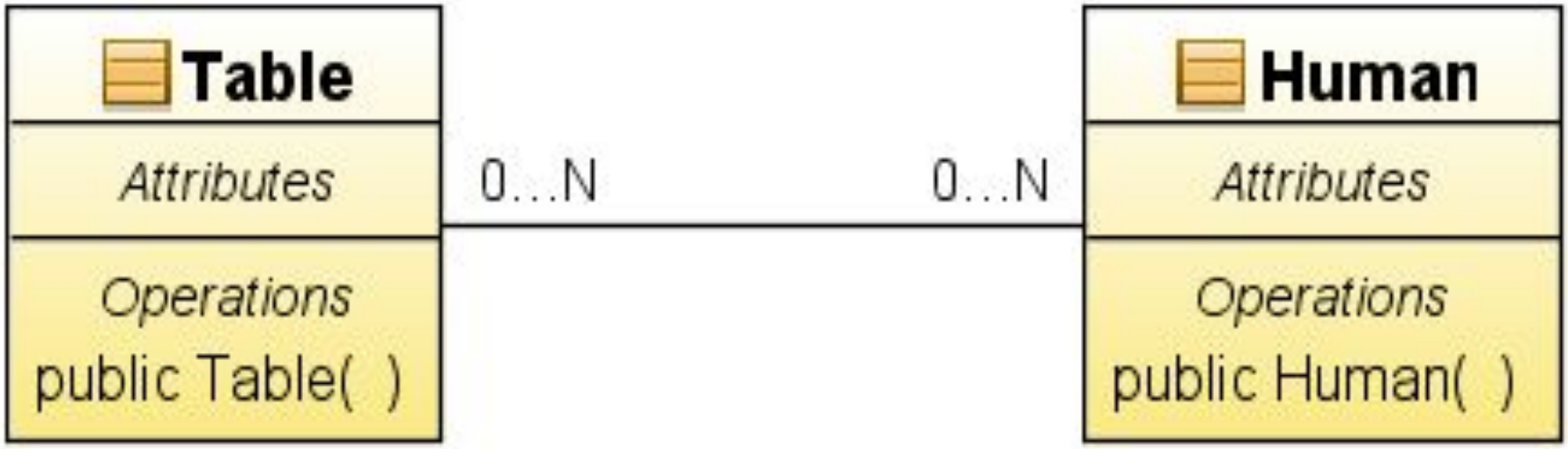
- Число объектов, участвующих в отношении, записывается рядом с именем роли. Запись «0…n» означает «от нуля до бесконечности». Приняты также обозначения:
  - «1…n» — от единицы до бесконечности;
  - «0» — ноль;
  - «1» — один;
  - «n» — фиксированное количество;
  - «0..1» — ноль или один.

Код

```
package ifmo.oop;

public class Group {

    private Student[] students;

    public Group() {
    }

    public void setStudents(Student[] students) {
        this.students = students;
    }

    public Student[] getStudents() {
        return students;
    }
}
```

Ассоциация

- Если объекты одного класса ссылаются на один или более объектов другого класса, но ни в ту, ни в другую сторону отношение между объектами не носит характера «владения», или контейнеризации, такое отношение называют **ассоциацией** (association).
- Отношение ассоциации изображается так же, как и отношение агрегации, но линия, связывающая классы, — простая, без ромбика.



Метаклассы

- Каждый класс всегда имеет строгий шаблон, задаваемый языком программирования или выбранной объектной моделью.
- Таким образом, класс можно рассматривать как объект, у которого есть свойства. Также класс может обладать поведением, то есть поддерживать методы. А раз для любого объекта существует шаблон, описывающий свойства и поведение этого объекта, значит, его можно определить и для класса.
- Такой шаблон, задающий различные классы, называется **метаклассом**.
- В языке Java также есть метакласс. Это класс, который так и называется — Class (описывает классы), он располагается в библиотеке java.lang.

Достоинства ООП

- Классы позволяют проводить конструирование из полезных компонентов, обладающих простыми инструментами, что позволяет абстрагироваться от деталей реализации.
- Данные и операции над ними образуют определенную сущность, и они не разносятся по всей программе, а описываются вместе. Локализация кода и данных улучшает наглядность и удобство сопровождения программного обеспечения.
- Инкапсуляция позволяет привнести свойство модульности, что облегчает распараллеливание выполнения задачи между несколькими исполнителями и обновление версий отдельных компонентов.

Достоинства ООП

- ООП дает возможность создавать расширяемые системы. Компоненты могут быть добавлены на этапе исполнения программы.
- Обработка разнородных структур данных. Программы могут работать, не различая вида объектов, что существенно упрощает код. Новые виды могут быть добавлены в любой момент.
- Изменение поведения во время исполнения. На этапе исполнения один объект может быть заменен другим, что позволяет легко, без изменения кода, адаптировать алгоритм в зависимости от того, какой используется объект.
- Реализация работы с наследниками. Алгоритмы можно обобщить настолько, что они уже смогут работать более чем с одним видом объектов.
- Создание «каркаса». Независимые от приложения части предметной области могут быть реализованы в виде набора универсальных классов, или каркаса (framework).

Достоинства ООП

- Сокращается время на разработку.
- Компоненты многоразового использования обычно содержат гораздо меньше ошибок, чем вновь разработанные.
- Когда некий компонент используется сразу несколькими клиентами, улучшения, вносимые в его код, одновременно оказывают положительное влияние и на множество работающих с ним программ.
- Если программа опирается на стандартные компоненты, ее структура и пользовательский интерфейс становятся более унифицированными.

Недостатки ООП

- Документирование классов — ресурсоёмкая задача.
- В сложных иерархиях классов поля и методы обычно наследуются с разных уровней. И не всегда легко определить, какие поля и методы фактически относятся к данному классу. Для получения такой информации нужны специальные инструменты, вроде навигаторов классов.
- Методы, как правило, короче процедур, зато их намного больше. В коротких методах легче разобраться, но они неудобны тем, что код для обработки сообщения иногда «размазан» по многим маленьким методам.

Недостатки ООП

- Злоупотребление инкапсуляцией данных. Чем больше логики и данных скрыто в недрах класса, тем сложнее его расширять.
- Неэффективность на этапе выполнения, в т.ч. инкапсуляция данных.
- Неэффективность в смысле распределения памяти.
- Излишняя универсальность.