**Наследование в Java и его преимущества**

***Наследование (inheritance)*** — механизм, который позволяет описать новый класс на основе существующего (родительского). При этом свойства и функциональность родительского класса заимствуются новым классом.

**Пример:**

**public** **class** Car {

**private** String model;

**private** **int** maxSpeed;

**private** **int** yearOfManufacture;

**public** Car(String model, **int** maxSpeed, **int** yearOfManufacture) {

**this**.model = model;

**this**.maxSpeed = maxSpeed;

**this**.yearOfManufacture = yearOfManufacture;

}

**public** **void** gas() {

//...газ

}

**public** **void** brake() {

//...тормоз

}

}

**public** **class** Truck **extends** Car {

**public** Truck(String model, **int** maxSpeed, **int** yearOfManufacture) {

**super**(model, maxSpeed, yearOfManufacture);

}

}

**public** **class** Sedan **extends** Car {

**public** Sedan(String model, **int** maxSpeed, **int** yearOfManufacture) {

**super**(model, maxSpeed, yearOfManufacture);

}

}

Есть некая программа, в рамках которой мы работаем с различными типами автомобилей. Даже если ты не автолюбитель, наверняка знаешь, что типов этих самых автомобилей на свете великое множество :) Поэтому общие свойства автомобилей выделяем в общий класс-родитель — Car. А что общего у всех автомобилей вне зависимости от типа? У любой машины есть год выпуска, название модели и максимальная скорость. Эти свойства выносим в поля model, maxSpeed, yearOfManufacture. Что касается поведения, любая машина может газовать и тормозить :) Это поведение мы определяем в методах gas() и brake(). Какие выгоды это нам дает? Прежде всего — сокращение объема кода. Конечно, можем обойтись и без родительского класса. Но поскольку каждая машина должна уметь газовать и тормозить, нам придется создавать методы gas() и brake() в классе Truck, в классе Sedan, в классе F1Car, в классе Sportcar и во всех остальных классах машин. Представь, сколько лишнего кода мы при этом напишем. Не забывай и о полях model, maxSpeed и yearOfManufacture: если откажемся от родительского класса, будем создавать их в каждом из классов-машин!

Когда у нас наберется пара десятков классов-машин, объем повторяющегося кода станет действительно серьезным. Вынесение общих полей и методов (еще говорят — «состояния» и «поведения») в класс-родитель позволит нам сэкономить кучу времени и места. Если же у какого-то типа есть свойства или методы, уникальные только для него и отсутствующие у других типов машин, — не беда. Их всегда можно создать в классе-потомке, отдельно от всех остальных.

**Пример:**

**public** **class** F1Car **extends** Car {

**public** **void** pitStop() {

//...пит-стоп делают только гоночные автомобили

}

**public** **static** **void** main(String[] args) {

F1Car formula1Car = **new** F1Car();

formula1Car.gas();

formula1Car.pitStop();

formula1Car.brake();

}

}

Возьмем случай с гоночными машинами Формулы-1. У них, в отличие от «сородичей», есть уникальное поведение — время от времени они заезжают на пит-стоп. Нам это не мешает. Общее поведение мы уже описали в родительском классе Car, а специфическое поведение классов-потомков можем добавить внутри классов.

Это касается и полей: если у дочернего класса есть уникальные свойства, спокойно объявляем эти поля внутри него и не переживаем :) **Возможность повторного использования кода — главное преимущество наследования.** Для программиста очень важно не писать лишний объем кода. Ты не раз столкнешься с этим в работе. Пожалуйста, запомни еще одну крайне важную вещь: **в Java нет множественного наследования.** Каждый класс наследуется только от одного класса. О причинах этого подробнее поговорим в будущих лекциях, пока просто запомни. Этим Java, кстати, отличается от некоторых других ООП-языков. Например, в С++ множественное наследование есть. С наследованием все более-менее ясно — идем дальше.

**Бебякин Марк**

**КИ-25**