**Наследование. Преимущество наследования**

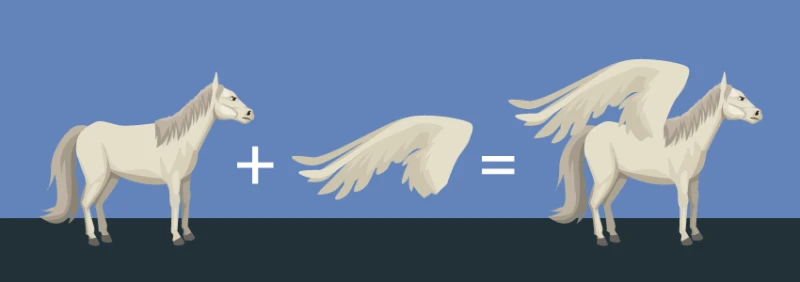
[Java Core](https://javarush.ru/quests/QUEST_JAVA_CORE)

[Уровень 1](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=1), Лекция 5

— Привет, Амиго! Сейчас будет одна тема, которой, я думаю, ты будешь частенько пользоваться. Это – наследование.

Программирование, для несведущих, неотличимо от магии. Поэтому начну с такой интересной аналогии…

Предположим, что ты волшебник и хочешь создать летающую лошадь. С одной стороны, ты бы мог попробовать наколдовать пегаса. Но т.к. пегасов в природе не существует, это будет очень непросто. Придется очень много делать самому. Куда проще взять лошадь и приколдовать ей крылья.

[](https://cdn.javarush.ru/images/article/9cdc9e25-4643-4d6a-bd3b-616ff3840aac/original.jpeg)

В программировании такой процесс называется «наследование». Предположим тебе нужно написать очень сложный класс. Писать с нуля долго, потом еще долго все тестировать и искать ошибки. Зачем идти самым сложным путем? Лучше поискать – а нет ли уже такого класса?

Предположим, ты нашел класс, который своими методами реализует 80% нужной тебе функциональности. Ты можешь просто скопировать его код в свой класс. Но у такого решения есть несколько минусов:

1) Найденный класс уже может быть скомпилирован в байт-код, а доступа к его исходному коду у тебя нет.

2) Исходный код класса есть, но ты работаешь в компании, которую могут засудить на пару миллиардов за использование даже 6 строчек чужого кода. А потом она засудит тебя.

3) Ненужное дублирование большого объема кода. Кроме того, если автор чужого класса найдет в нем ошибку и исправит ее, у тебя эта ошибка останется.

Есть решение потоньше, и без необходимости получать легальный доступ к коду оригинального класса. В Java ты можешь просто объявить тот класс родителем твоего класса. Это будет эквивалентно тому, что ты добавил код того класса в код своего. В твоем классе появятся все данные и все методы класса-родителя. Например, можно делать так: наследуемся от «лошади», добавляем «крылья» — получаем «пегаса»

[](https://cdn.javarush.ru/images/article/57aaa0cc-a622-447f-8070-3fe93ce1d839/original.jpeg)

— Очень интересно, продолжай.

— Наследование можно использовать и для других целей. Допустим, у тебя есть десять классов, которые очень похожи, имеют совпадающие данные и методы. Ты можешь создать специальный базовый класс, вынести эти данные (и работающие с ними методы) в этот базовый класс и объявить те десять классов его наследниками. Т.е. указать в каждом классе, что у него есть класс-родитель – данный базовый класс.

Также как преимущества абстракции раскрываются только рядом с инкапсуляцией, так и преимущества наследования гораздо сильнее при использовании полиморфизма. Но о нем я расскажу завтра. Сегодня же мы рассмотрим несколько примеров использования наследования.

Предположим, мы пишем программу, которая играет в шахматы с пользователем, тогда нам понадобятся классы для фигур. Какие бы ты предложил классы, Амиго?

— Король, Ферзь, Слон, Конь, Ладья и Пешка.

— Отлично. Ничего не упустил.

— А какие бы данные ты предложил хранить в этих классах?

— Координаты x и y, а также ее ценность (worth). Ведь некоторые фигуры ценнее других.

— А в чем отличия этих классов?

— Отличия в том, как они ходят, фигуры. В поведении.

— Да. Вот как можно было бы описать их в виде классов

|  |  |  |
| --- | --- | --- |
| class King  {  int x;  int y;  int worth;  void kingMove()  {  //код, решающий,  //как пойдет король  }  } | class Queen  {  int x;  int y;  int worth;  void queenMove()  {  //код, решающий,  //как пойдет ферзь  }  } | class Rook  {  int x;  int y;  int worth;  void rookMove()  {  //код, решающий,  //как пойдет ладья  }  } |
| class Knight  {  int x;  int y;  int worth;  void knightMove()  {  //код, решающий,  //как пойдет конь  }  } | class Bishop  {  int x;  int y;  int worth;  void bishopMove()  {  //код, решающий,  //как пойдет слон  }  } | class Pawn  {  int x;  int y;  int worth;  void pawnMove()  {  //код, решающий,  //как пойдет пешка  }  } |

3

Задача

Java Core,  1 уровень,  5 лекция

Набираем код Ӏ Java Core: 1 уровень, 5 лекция

Java Core: 1 уровень, 5 лекция. Иногда думать не надо, строчить надо! Как ни парадоксально звучит, порой пальцы «запоминают» лучше, чем сознание. Вот почему во время обучения в секретном центре JavaRush вы иногда встречаете задания на набор кода. Набирая код, вы привыкаете к синтаксису и зарабатываете немного материи. А ещё — боретесь с ленью.

— Да, именно так я бы и написал.

— А вот, как можно было бы сократить код с помощью наследования. Мы могли бы вынести одинаковые методы и данные в общий класс. Назовем его ChessItem. Объекты класса ChessItem не имеет смысла создавать, так как ему не соответствует ни одна шахматная фигура, но от него было бы много пользы:

|  |  |  |
| --- | --- | --- |
| class King extends ChessItem  {  void kingMove()  {  //код, решающий,  //как пойдет король  }  } | class Queen extends ChessItem  {  void queenMove()  {  //код, решающий,  //как пойдет ферзь  }  } | class Rook extends ChessItem  {  void rookMove()  {  //код, решающий,  //как пойдет ладья  }  } |
|  | class ChessItem  {  int x;  int y;  int worth;  } |  |
| class Knight extends ChessItem  {  void knightMove()  {  //код, решающий,  //как пойдет конь  }  } | class Bishop extends ChessItem  {  void bishopMove()  {  //код, решающий,  //как пойдет слон  }  } | class Pawn extends ChessItem  {  void pawnMove()  {  //код, решающий,  //как пойдет пешка  }  } |

3

Задача

Java Core,  1 уровень,  5 лекция

Набираем больше кода Ӏ Java Core: 1 уровень, 5 лекция

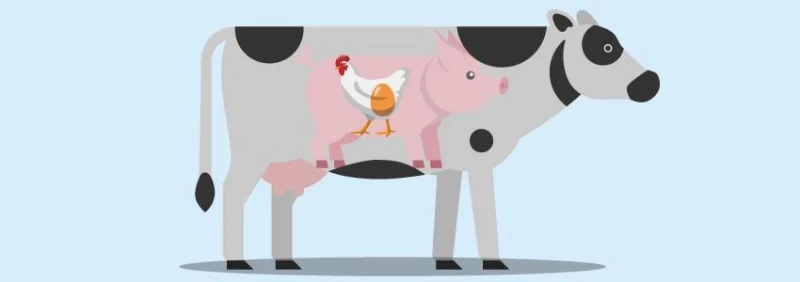
Java Core: 1 уровень, 5 лекция. Внимание! Объявляется набор кода на JavaRush. Для этого включите режим повышенной внимательности, расслабьте пальцы, читайте код и… набирайте его в соответствующем окошке. Набор кода — вовсе не бесполезное занятие, как может показаться на первый взгляд: благодаря ему новичок привыкает к синтаксису и запоминает его (современные IDE редко дают ему это сделать).

— Как интересно.

— Именно! Особенно много преимуществ мы получаем, когда в проекте тысячи различных объектов и сотни классов. Тогда правильно подобранными классами можно не только существенно упростить логику, но и сократить код в десятки раз.

— А что нужно чтобы унаследовать какой-то класс?

— Для этого после объявления нашего класса нужно указать ключевое слово extends и написать имя родительского класса. Унаследоваться можно только от одного класса.

[](https://cdn.javarush.ru/images/article/6fdefd74-6078-4fc1-bf1a-10298e89506c/original.jpeg)

На картинке мы видим «корову», унаследованную от «свиньи». «Свинья» унаследована от «курицы», «курица» от «яйца». Только один родитель! Такое наследование не всегда логично. Но если есть только свинья, а очень нужна корова, программист зачастую не может устоять перед желанием сделать «корову» из «свиньи».

— А если мне хочется унаследоваться от двух классов. Можно же что-то сделать?!

— Почти ничего. Множественного наследования классов в Java нет: класс может иметь только одного класса-родителя. Но есть множественное наследование интерфейсов. Это немного снижает остроту проблемы.

— Ясно. А что такое интерфейс?

— Про интерфейсы я расскажу тебе завтра, а пока давай продолжим разбираться с наследованием.