<https://javarush.ru/groups/posts/2018-kak-rabotaet-operator-instanceof>

Привет! Сегодня мы поговорим об операторе instanceof, рассмотрим примеры его использования и затронем некоторые связанные с его работой моменты :) На ранних уровнях JavaRush ты уже сталкивался с этим оператором. Помнишь, зачем он нужен? Если нет — не беда, давай вспомним вместе. **Оператор instanceof нужен, чтобы проверить, был ли объект, на который ссылается переменная X, создан на основе какого-либо класса Y.** Звучит просто. Почему же мы вернулись к этой теме? Прежде всего потому, что теперь ты хорошо знаком с механизмом наследования в Java и остальными принципами ООП. Тема **instanceof** будет гораздо понятнее, и мы рассмотрим более продвинутые примеры использования. Поехали! Ты наверняка помнишь, что оператор instanceof возвращает значение true, если проверка показала истинность, или false, если результат был ложным. Следовательно, чаще всего он встречается в разного рода проверочных условиях (if…else). Начнем с примеров попроще:

**public** **class** Main {

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

Integer x = **new** Integer(22);

System.out.println(x **instanceof** Integer);

}

}

Как думаешь, что будет выведено в консоль? Ну, здесь это очевидно :) Объект **х** является Integer, поэтому результатом будет true. Вывод в консоль: true Попробуем проверить его на принадлежность, например, к String:

**public** **class** Main {

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

Integer x = **new** Integer(22);

System.out.println(x **instanceof** String);// ошибка!

}

}

Мы получили ошибку. Причем обрати внимание: компилятор выдал ее еще до выполнения кода! Он сразу увидел, что Integer и String не могут быть автоматически преобразованы друг к другу и не состоят в связях наследования. Следовательно, объект класса Integer не создастся на основе String. Это удобно и помогает избежать странных ошибок уже во время выполнения программы, так что тут компилятор нас выручил :) Теперь давай попробуем рассмотреть примеры посложнее. Раз уж мы упомянули наследование, поработаем вот с такой небольшой системой классов:

**public** **class** Animal {

}

**public** **class** Cat **extends** Animal {

}

**public** **class** MaineCoon **extends** Cat {

}

Мы уже знаем, как ведет себя instanceof, когда мы проверяем принадлежность объекта к какому-то классу в обычной ситуации, но что будет, если мы добавим сюда отношение «родитель-потомок»? Например, как думаешь, что выдаст вот такая проверка:

**public** **class** Main {

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

Cat cat = **new** Cat();

System.out.println(cat **instanceof** Animal);

System.out.println(cat **instanceof** MaineCoon);

}

}

Вывод: true false Главный вопрос, на который нужно ответить, — как именно instanceof расшифровывает понятие «объект создан на основе класса»? У нас в результате получилось Сat instanceof Animal = true, но ведь к такой формулировке можно и придраться. Почему это объект Cat создается на основе класса Animal? Разве он не создается только на основе собственного класса? Ответ достаточно прост, и ты, возможно, уже додумался до него. Вспомни порядок вызова конструкторов и инициализации переменных при создании объекта. Мы уже рассматривали эту тему в [статье о конструкторе классов](https://javarush.ru/groups/posts/1927-konstruktorih-bazovihkh-klassov--). Вот пример из той лекции:

**public** **class** Animal {

String brain = "Изначальное значение brain в классе Animal";

String heart = "Изначальное значение heart в классе Animal";

**public** **static** **int** animalCount = 7700000;

**public** Animal(String brain, String heart) {

System.out.println("Выполняется конструктор базового класса Animal");

System.out.println("Были ли уже проинициализированы переменные класса Animal?");

System.out.println("Текущее значение статической переменной animalCount = " + animalCount);

System.out.println("Текущее значение brain в классе Animal = " + **this**.brain);

System.out.println("Текущее значение heart в классе Animal = " + **this**.heart);

**this**.brain = brain;

**this**.heart = heart;

System.out.println("Конструктор базового класса Animal завершил работу!");

System.out.println("Текущее значение brain = " + **this**.brain);

System.out.println("Текущее значение heart = " + **this**.heart);

}

}

**class** Cat **extends** Animal {

String tail = "Изначальное значение tail в классе Cat";

**static** **int** catsCount = 37;

**public** Cat(String brain, String heart, String tail) {

**super**(brain, heart);

System.out.println("Конструктор класса Cat начал работу (конструктор Animal уже был выполнен)");

System.out.println("Текущее значение статической переменной catsCount = " + catsCount);

System.out.println("Текущее значение tail = " + **this**.tail);

**this**.tail = tail;

System.out.println("Текущее значение tail = " + **this**.tail);

}

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

Cat cat = **new** Cat("Мозг", "Сердце", "Хвост");

}

}

И если ты его запустишь в IDE, вывод в консоль будет выглядеть так: Выполняется конструктор базового класса Animal Были ли уже проинициализированы переменные класса Animal? Текущее значение статической переменной animalCount = 7700000 Текущее значение brain в классе Animal = Изначальное значение brain в классе Animal Текущее значение heart в классе Animal = Изначальное значение heart в классе Animal Конструктор базового класса Animal завершил работу! Текущее значение brain = Мозг Текущее значение heart = Сердце Конструктор класса Cat начал работу (конструктор Animal уже был выполнен) Текущее значение статической переменной catsCount = 37 Текущее значение tail = Изначальное значение tail в классе Cat Текущее значение tail = Хвост Теперь вспомнил? :) Конструктор базового класса, если он есть, всегда вызывается первым при создании любого объекта. Instanceof руководствуется именно этим принципом, когда пытается определить, был ли объект А создан на основе класса Б. Если конструктор базового класса вызван, значит никаких сомнений быть не может. Со второй проверкой все проще:

System.out.println(cat **instanceof** MaineCoon);

Конструктор MaineCoon не вызывался при создании Cat, что логично. Ведь MaineCoon — потомок Cat, а не предок. Но шаблоном для Cat он не является. Ок, с этим вроде понятно. А что будет, если мы сделаем вот так:

**public** **class** Main {

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

Cat cat = **new** MaineCoon();

System.out.println(cat **instanceof** Cat);

System.out.println(cat **instanceof** MaineCoon);

}

}

Хм...тут уже посложнее. Давай попробуем порассуждать. У нас есть переменная типа Cat, и ей мы присвоили объект типа MaineCoon. Кстати, а почему это вообще работает? Разве так можно делать? Можно. Ведь любой мейн-кун — это кошка. Если не совсем понятно, вспомни пример с расширением примитивных типов:

**public** **class** Main {

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

**long** x = 1024;

}

}

Число 1024 — это short: он легко помещается в переменную long, ведь по количеству байт для него достаточно места (помнишь пример с матрешками?). Объект-потомок всегда можно присвоить в переменную-предка. Пока просто запомни это, а в следующих лекциях мы еще разберем этот процесс. Так что же выведет наш пример?

Cat cat = **new** MaineCoon();

System.out.println(cat **instanceof** Cat);

System.out.println(cat **instanceof** MaineCoon);

Что будет проверять instanceof: нашу переменную класса Cat или наш объект класса MaineCoon? На самом деле, ответить на этот вопрос просто. Нужно всего лишь еще раз прочитать определение нашего оператора: Оператор instanceof нужен для того, чтобы проверить, был ли объект, на которую ссылается переменная X, создан на основе какого-либо класса Y. **Оператор instanceof проверяет именно происхождение объекта, а не переменной.** Поэтому в примере оба раза в консоли выведет **true**: у нас объект типа MaineCoon. Естественно, он был создан на основе класса MaineCoon, но и на основе родительского класса Cat тоже!

[undefined](https://javarush.ru/authors/null)

[Professor Hans Noodles](https://javarush.ru/authors/null)

**Комментарии (101)**

**ЧТОБЫ ПОСМОТРЕТЬ ВСЕ КОММЕНТАРИИ ИЛИ ОСТАВИТЬ КОММЕНТАРИЙ,**[**ПЕРЕЙДИТЕ В ПОЛНУЮ ВЕРСИЮ**](https://javarush.ru/groups/posts/2018-kak-rabotaet-operator-instanceof?post=full#discussion)

[**Grock**](https://javarush.ru/users/3103172) Уровень 25

24 сентября 2022

И еще момент. Если мы в классе-потомке переопределили метод родительского класса, но хотим вызвать метод родительского класса, то этого можно добиться, добавив слово super с точкой перед вызовом метода родительского класса, например:

**super**.animal\_1();

, где animal\_1() - имя переопределенного метода.

[**Grock**](https://javarush.ru/users/3103172) Уровень 25

24 сентября 2022

Для наглядности:

**public** **class** Test {

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

Animal catExtAnimal = **new** Cat();

Cat cat = **new** Cat();

//Если тип объекта Animal (родитель) а сам объект - наследник (Cat), доступны только методы родителя

catExtAnimal.animal\_1(); //Метод Cat\_1() не доступен для animal\_cat.

//Вывод (консоль): Метод animal (переопределен в Cat).

//Если метод родителя переопределен в классе-наследнике,

//вызывается переопределенный методкласса-наследника.

catExtAnimal.cat\_1(); //Ошибка. Не удается найти метод cat\_1() в классе-родителе (Animal)

//Если тип объекта и объект одинаковы, доступны все методы (родителя и наследника)

cat.animal\_1(); //Для объекта (Cat) доступен метод его родителя (Animal).

//Вывод (консоль): Метод animal (переопределен в Cat).

//Если метод родителя переопределен в классе-наследнике,

//вызывается переопределенный методкласса-наследника.

cat.cat\_1(); //Для объекта (Cat) доступен метод его собственного класса.

// Вывод (консоль): Метод cat

}

**static** **class** Animal {

**public** **void** animal\_1() {

System.out.println("Метод animal");}}

**static** **class** Cat **extends** Animal {

**public** **void** cat\_1() {

System.out.println("Метод cat");}

@Override

**public** **void** animal\_1() {

System.out.println("Метод animal (переопределен в Cat)");}}

}

**Вывод в консоль:** (если не вызывать метод с ошибкой catExtAnimal.Cat\_1();)

Метод animal (переопределен в Cat)

Метод animal (переопределен в Cat)

Метод cat

[**Игорь Зваричук**](https://javarush.ru/users/2906367) Уровень 30, Москва, Russian Federation

29 августа 2022

Спасибо вам! :)

[**Ada**](https://javarush.ru/users/3038683) Уровень 46

19 мая 2022

Лекция супер! 🔥

[**Стас**](https://javarush.ru/users/2469321) Уровень 19, Москва

27 апреля 2022

помогите разобраться. при создании переменной типа Cat cat = new Meincoon(); т.е. у меня переменная будет типа Cat(т.е. все переменные и методы класса Cat будут доступны?) и в ней сохранен объект другого класса. что фактически можно делать с переменной Cat? будут ли там методы и переменные обоих классов или... ? вот это не понятно.

[**Rylero**](https://javarush.ru/users/2865180) Уровень 32, Москва, Russian Federation

5 апреля 2022

Не понял каким образом оператор instanceof проверяет какой конструктор был вызван - где-то сохраняется лог создания объекта?

[**Михаил**](https://javarush.ru/users/2814008) Уровень 32, Волжский, Russian Federation

8 ноября 2021

Не понял почему не компилируется пример из начала

System.out.println(x **instanceof** String);// ошибка!

Объясняют, что компилятор уже видит не возможность приведения типов. Но я и не делают приведение типов

x = (String) x;

а просто хочу узнать является ли он объектом этого класса. Почему он просто не выведет в консоль "false", как например в следующем примере

System.out.println(cat **instanceof** MaineCoon);

[**Александр Горохов**](https://javarush.ru/users/2385417) Уровень 28, Дятьково, Россия

27 сентября 2021

Как же надоело это "Перейдите в полную версию"

[**hidden #2595317**](https://javarush.ru/users/2595317) Уровень 45

17 июня 2021

"Ведь MaineCoon — потомок Cat, а не предок. Но шаблоном для Cat он не является." Стоит НО заменить на ПОЭТОМУ.

[**🦔 Виктор**](https://javarush.ru/users/2275393) Уровень 20, Москва, Россия **EXPERT**

26 декабря 2020

Спасибо за труды, но мне кажется, что зря статью смешали с примером из конструкторов, лучше бы взяли что-нибудь тематическое для наглядности. Закрепляем тему соседней статьёй [instanceof и основы наследования](https://javarush.ru/groups/posts/1947-instanceof-i-osnovih-nasledovanija). **Всё получится!**