# Лекция 8. Пакет java.lang

# Пакет java.lang

**java.lang (lang – сокращение от англ. language) – основной (базовый) пакет языка программирования Java.** Все классы в Java неявным образом импортируют классы этого пакета.

*(!) Соответственно, не нужно писать оператор импорта («import java.lang.\*;») для определения области видимости классов из данного пространства имен.*

Основу пакета составляют следующие классы и интерфейсы:

* ***java.lang.Object*** является прародителем всех классов в Java и стоит в вершине иерархии наследования классов Java. Любой класс, у которого явно не задан родитель, наследует по умолчанию классу Object.
* ***java.lang.Class*** – объектное представление сущности класса. Экземпляры типа Class – это описанные в исходном коде классы и интерфейсы, но уже в момент исполнения Java-программы. Загрузка объектов типа Class происходит на этапе запуска приложения. Необходимы данные объекты для создания экземпляров описанных типов в исходном коде.
* ***java.lang.String*** – класс, предоставляющий набор методов для работы со строками.
* ***java.lang.StringBuffer/java.lang.StringBuilder*** - классы используется для работы (создания и изменения) строк. Предоставляют оптимизированные способы работы со строками.
* ***java.lang.Number*** – абстрактный класс, являющийся суперклассом для классов-оберток числовых примитивных типов Java.
* ***java.lang.Character*** – класс-обертка для примитивного типа char.
* ***java.lang.Boolean*** – класс-обертка для типа boolean.
* ***java.lang.Math*** – предоставляет набор методов для выполнения ряда математических функций.
* ***java.lang.Throwable*** – базовый класс для объектов, представляющих исключения.
* ***java.lang.Thread*** – представляет собой вычислительный поток в Java. Виртуальная машина Java позволяет одновременно выполнять несколько потоков.
* ***java.lang.System*** – содержит поля и методы для работы системного уровня (текущее системное время, информацию о потоке, в котором выполняется код, потоки ввода и вывода данных и т.д.).
* ***java.lang.Cloneable*** – интерфейс, который должен быть реализован классом, объекты которого могут быть клонированы с помощью средств JVM.
* ***java.lang.Runnable*** – интерфейс, который должен быть реализован классом, набор инструкций которого должен быть выполнен в отдельном потоке.
* ***java.lang.Comparable*** – интерфейс, реализация которого позволяет упорядочивать, сортировать и сравнивать объекты

# Класс java.lang.Object

Класс Object является базовым для всех остальных классов в Java. В нем содержится набор методов, которые поддерживаются любым классом в Java.

***public boolean equals(Object obj)*** метод определения, являются ли объекты одинаковыми. Оператор сравнения («==») определяет равенство объектных ссылок, тогда как метод ***equals()*** определяет, содержат ли объекты одинаковую информацию, иными словами, находятся ли они в одинаковом состоянии. Реализация этого метода в классе Object такова, что значение ***true*** будет возвращено тогда и только тогда, если объектные ссылки переданного объекта и объекта, у которого вызывается метод совпадают, то есть в исходниках Java метод выглядит так:

|  |
| --- |
| public boolean equals(Object obj)  {  return (this == obj);  } |

В классах-наследниках, этот метод может быть переопределен, чтобы отражать действительное равенство объектов. Например, сравнение объектов-оберток целых чисел (класс Integer), должно, по логике, возвращать значение true, если равны значения **int** чисел, которые обернуты, даже если различаются объектные ссылки на сами объекты класса-обертки Integer.

Метод ***equals()*** может быть переопределен любым способом (например, всегда возвращает false, или, наоборот, - true) - компилятор, конечно же, не будет проводить анализ реализации и давать рекомендации, однако существует ряд рекомендаций по реализации метода ***equals()***:

1. **рефлексивность**: для любой объектной ссылки x, вызов x.equals(x) возвращает true.
2. **симметричность**: для любых объектных ссылок x и y, вызов x.equals(y) возвращает true тогда и только тогда, если вызов y.equals(x) возвращает true.
3. **транзитивность**: для любых объектных ссылок x, y и z, если x.equals(y) возвращает true и y.equals(z) возвращает true, тогда вызов x.equals(z) должен вернуть true
4. **непротиворечивость**: для любых объектных ссылок x и y, многократные последовательные вызовы x.equals(y) возвращают одно и то же значение (либо всегда true либо всегда false)
5. для любой не равной null объектной ссылки x, вызов x.equals(null) должен вернуть значение false.

|  |
| --- |
| **package** com.tsystems.entity;  **public** **class** Rectangle  {  **public** **int** sideA;  **public** **int** sideB;  **public** Rectangle(**int** x, **int** y)  {  sideA=x;  sideB=y;  }  **public** **boolean** equals(Object obj)  {  **if** (obj == **null**)  **return** **false**;  **if** (**this** == obj)  **return** **true**;  **if** (obj.getClass() != **this**.getClass())  **return** **false**;  Rectangle ref=(Rectangle) obj;  **return** ((**this**.sideA == ref.sideA) && (**this**.sideB == ref.sideB));  }  @Override **public** String toString()  {  **return** "sideA=" + sideA + "; sideB=" + sideB;  }  **public** **static** **void** main(String[] args)  {  Rectangle r1=**new** Rectangle(**10**, **20**);  Rectangle r2=**new** Rectangle(**10**, **20**);  Rectangle r3=**new** Rectangle(**20**, **10**);  System.***out***.println("r1: " + r1);  System.***out***.println("r2: " + r2);  System.***out***.println("r3: " + r3);  // compare r1 with r1  System.***out***.println("r1 == r1 : " + (r1 == r1));  System.***out***.println("r1.equals(r1) : " + r1.equals(r1));  // compare r1 with r2  System.***out***.println("r1 == r2 : " + (r1 == r2));  System.***out***.println("r1.equals(r2) == " + r1.equals(r2));  // compare r2 with r3  System.***out***.println("r2 == r3 : " + (r2 == r3));  System.***out***.println("r1.equals(null) == " + r2.equals(**null**));  }  } |

Пример выше иллюстрирует работу метода equals().

***public int hashCode()*** – возвращает хеш-код (hash code) объекта. Хеш-код - это целое число, которое с очень большой вероятностью является уникальным для данного объекта. Можно пользоваться стандартной реализацией, но, если будет принято решение изменить метод вычисления хеш-кода, то необходимо убедиться, что в этом случае будет возвращаться одно и то же значение для равных между собой объектов. То есть, если x.equals(y) возвращает true, то хеш-коды x и y должны совпадать, то есть вызовы x.hashCode() и y.hashCode() должны возвращать одно и то же целое значение. Формально правила, которым должна следовать реализация метода, формулируется следующим образом:

1. в одном запуске программы, для одного объекта, при вызове метода hashCode(), должно возвращаться одно и то же int значение, если между этими вызовами НЕ были затронуты данные, используемые для проверки объектов на идентичность в методе equals(). Это число НЕ обязано быть одним и тем же при повторном запуске той же программы, даже если все данные будут идентичны
2. если два объекта идентичны, то есть вызов метода equals(Object) возвращает true, тогда вызов метода hashCode() у каждого из этих двух объектов должен возвращать одно и то же значение
3. если два объекта различны, то есть вызов метода equals(Object) возвращает false, тогда различие их хеш-кодов желательно, но НЕ обязательно.

В классе Object реализация метода hashCode() использует для получения результата преобразование внутреннего адреса объекта в памяти, поэтому если не перекрывать эту реализацию, то для разных объектов будут возвращены различные значения.

***public String toString()*** возвращает строковое представление объекта. В классе Object этот метод реализован следующим образом:

|  |
| --- |
| public String toString()  {  return getClass().getName() + "@" + Integer.toHexString(hashCode());  } |

То есть метод возвращает строку, содержащую название класса объекта и его хеш-код. В классах-наследниках этот метод может быть переопределен для получения более наглядного пользовательского представления объекта. Обычно это значения некоторых полей, характеризующих экземпляр. Например, для книги это может быть название, автор и количество страниц:

|  |
| --- |
| **public** **class** Book  {  **private** String title;  **private** String author;  **private** **int** pages;  **public** Book(String title, String author, **int** pagesNumber)  {  **super**(); //вызов констурктора класса Object  **this**.title=title;  **this**.author=author;  **this**.pages=pagesNumber;  }  **public** String toString()  {  **return** "Book: " + title + " ( " + author + ", " + pages + " pages )";  }  **public** **static** **void** main(String[] args)  {  Book book=**new** Book("Java2", "Sun", **1000**);  System.***out***.println("object is: " + book);  }  } |
|  |

Для получения строкового представления объекта book, был вызван его метод toString() (оператор «+» определен для строк таким образом, что все операнды будут сначала приведены к String, для объектов в этом случае вызывается метод toString()).

***wait(), notify(), notifyAll()*** – методы поддержания многопоточности. Определены, как final, соответственно, НЕ могут быть переопределены в классах-наследниках.

***protected void finalize() throws Throwable*** – метод удаления объекта. Вызывается Java-машиной перед тем, как garbage collection (сборщик мусора) очистит память, занимаемую объектом. Объект считается пригодным для сборщика мусора, когда выполняющейся части программы не будет доступно ни одной ссылки на объект. Перед очисткой занимаемой объектом памяти, будет произведен вызов его метода finalize(). Реализация этого метода в классе Object не производит никаких действий. В классах-наследниках этот метод может быть переопределен для проведения всех необходимых действий по освобождению различных занимаемых ресурсов (закрытия сетевых соединений, файлов и т.д.).

***protected native Object clone() throws CloneNotSupportedException*** – метод создания копии объекта. Для того, чтобы этот метод можно было вызывать, класс должен реализовывать интерфейс ***Cloneable***. Этот интерфейс не определяет никаких методов, т.е. является «маркером», и определяет, что можно создавать копии объектов этого класса. В классе Object метод clone() реализован таким образом, что будут скопированы только базовые типы и ссылки на объекты. Если же потребуется «глубокое» копирование, то есть скопировать не только ссылки на объекты, но и создать копии объектов - в классах-наследниках метод clone() можно переопределить.