

Equals

По умолчанию equals **идентичен** “==”:

- возвращает true, когда переменные ссылались на один и тот же объект.

Методы equals и toString.

Если вы перезаписываете метод equals, всегда используйте в его параметре тип данных Object.

Правильно и логично перезаписанный метод equals должен обладать следующими свойствами:

1. Симметричность — для non-null ссылочных переменных a и b, a.equals(b) возвращает true тогда и только тогда, когда b.equals(a) возвращает true;
2. Рефлексивность — для non-null ссылочной переменной a, a.equals(a) всегда должно возвращать true;
3. Транзитивность — для non-null ссылочных переменных a, b и c, если a.equals(b) и b.equals(c) возвращает true, то a.equals(c) тоже должно возвращать true;
4. Постоянство — для non-null ссылочных переменных a и b, неоднократный вызов a.equals(b) должен возвращать или только true, или только false;
5. Для non-null ссылочной переменной a, a.equals(null) всегда должно возвращать false;

Метод toString принадлежит классу Object, возвращает строковое представление объекта. Дефолтная реализация данного метода возвращает имя класса, @, число (результат метода hashCode данного объекта).

HashCode

Необходим для HashMap<> и HashSet<>

Поиск элемента и сравнение идет сначала по hashCode, а потом уже проводится проверка equals.

hashCode работает намного быстрее equals.

Методы equals и hashCode

Если Вы переопределили equals, то переопределите и hashCode.

Результат нескольких выполнений метода hashCode для одного и того же объекта должен быть одинаковым.

Если, согласно методу equals, два объекта равны, то и hashCode данных объектов обязательно должен быть одинаковым.

Если, согласно методу equals, два объекта НЕ равны, то hashCode данных объектов НЕ обязательно должен быть разным.

Ситуация, когда результат метода hashCode для разных объектов одинаков, называется коллизией. Чем её меньше, тем лучше.

Generics

Type – type place holder (заполнитель типа)

T не может быть static

```
class Info<T> {  
    private T value;  
  
    public Info(T t) {  
        this.value = t;  
    }  
  
    public String toString() {  
        return "{" + value + "}";  
    }  
  
    public T getValue() {  
        return value;  
    }  
}
```

Type erasure:

Невозможно создать два метода

```
public void abc(Info<String> info) {  
    String s = info.getValue();  
}  
  
public void abc(Info<Integer> info) {  
    int i = info.getValue();  
}  
  
// Потому что в JVM оба метода имеют  
// одинаковую сигнатуру public void abc(Info info)
```

Wild card

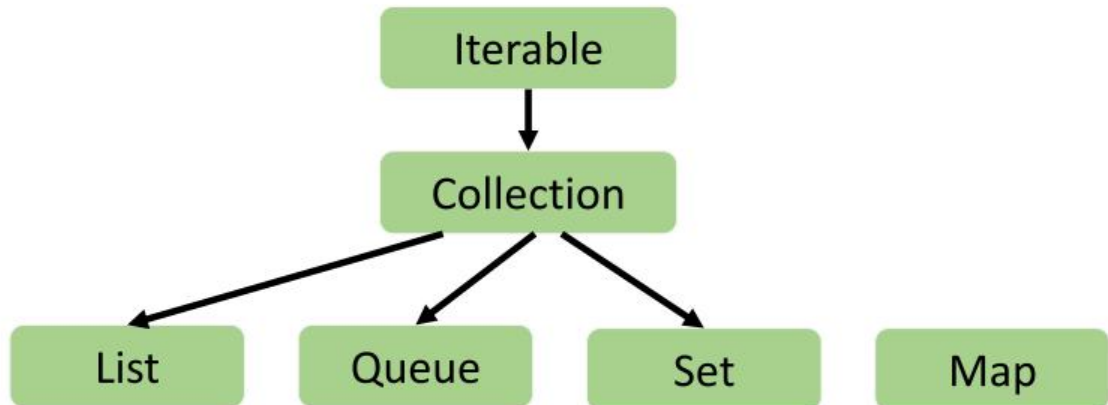
Нельзя добавить элемент в список с неизвестным типом элементов

```
List<?> list = new ArrayList<String>();  
list.add("hello");
```

Нужно использовать bounded wild card

Collection

Иерархия коллекций



LinkedList<>

Связанный список

Doubly LinkedList<> начинает поиск с конца списка

ListIterator<>

Методы класса Collections

binarySearch

Vector<>

Устаревший класс, похож на ArrayList<>, только многопоточный

Stack<>

Устаревший synchronized класс. Использует принцип LIFO.

Не рекомендован для использования

Push() – вставляет

Pop() - вытаскивает

HashCode and Equals

Объекты равны, если hashCode одинаковые и equals одинаковые

При создание HashMap<> Задаются параметры:

Initial capacity – начальный размер массива

Load factor – процент, после которого создается новый массив

TreeMap<>

LinkedHashMap<>

HashTable<>

Является synchronized классом

В Hashtable<> ни ключ ни значение не может быть null

CuncurrentHashMap<>

Лучше всего использовать для многопоточности

Deque and ArrayDeque

Двунаправленная очередь

Nested Classes

```
public class Test {  
    // статический класс  
    static class A {  
  
    }  
  
    // простой внутренний класс  
    class B {  
    }  
  
    void method() {  
        // локальный класс  
        class C {  
  
        }  
    }  
}
```

Static nested class

Inner class

Local inner class

Multithreading

Concurrency / Parallelism

Concurrency – согласованность (выполнение нескольких задач, но не обязательно в одно и то же время)

Concurrent – параллельный

- 1) Петь и есть (concurrency, одноядерный процессор)
- 2) Готовить и говорить по телефону (concurrency, многоядерный процессор) достигается за счет параллелизма.

Asynchronous / Synchronous

В синхронном программировании все задания выполняются друг за другом.

1) Написать два письма (делаются синхронно, одно за другим)
Сначала пишу 1е письмо.
Потом пишу 2е письмо.

2) Сделать бутерброд и постирать белье (делаются асинхронно).
Белье стирается, а в это время я ем бутерброд.

Volatile – данная переменная хранится только в основной памяти и не кэшируется в память ядер.

Volatile работает корректно, если только один поток ее изменяет, а все остальные читают.

Data race и synchronized methods

Понятие «монитор» и synchronized

Синхронизация происходит через монитор объекта или класса.

Если идет синхронизация по нестатическому методу, то всегда идет синхронизация по объекту, на котором вызывается данный метод (this не пишется).

При использовании блока синхронизации this пишется явно.

При использовании синхронизации на статическом методе, используется монитор всего класса (пишется synchronized MyClass.class в блоке синхронизации).

Синхронизация нескольких методов по одному объекту:

Методы wait и notify:

Извещение потоков о результатах выполнения других потоков.

Wait – ждать. Освобождает монитор и переводит вызывающий поток в состояние ожидания до тех пор, пока другой поток не вызовет метод notify().

Notify – будит другой поток, у которого до этого был вызван wait.

NotifyAll – будит все потоки.

Wait и notify вызывается на том объекте, на котором происходит синхронизация.

wait(1000) – поток будет ждать максимум 1 секунду, если его не разбудят.

Для проверки условий в потоке используется while, потому что поток может проснуться и без notify, а while всегда проверяет условия его выполнения, в отличие от if.

Для извещения потоком других потоков о своих действиях часто используются следующие методы:

wait - освобождает монитор и переводит вызывающий поток в состояние ожидания до тех пор, пока другой поток не вызовет метод notify();

notify – НЕ освобождает монитор и будит поток, у которого ранее был вызван метод wait();

notifyAll – НЕ освобождает монитор и будит все потоки, у которых ранее был вызван метод wait();

Возможные ситуации в многопоточном программировании

Deadlock – два потока заблокированы друг другом.

Чтобы не возникало такого, необходимо производить захват lock -ов в одном и том же порядке.

Livelock – два потока работают, только без особого прогресса. (Машины – роботы на мосту, которые уступают дорогу друг другу).

Lock starvation – менее приоритетные потоки очень долго ждут, чтобы запуститься.

Lock и ReentrantLock

lock()

unlock()

trylock() – поток пытается поставить Lock.

Даemon потоки

Это потоки,

setDaemon() – нужно вызывать после того как поток был создан и перед тем, как он был запущен

Прерывание потоков

isInterrupted()

Thread pool и ExecutorService

// Создание threadPool на один поток

```
ExecutorService executorServiceSingleton = Executors.newSingleThreadExecutor();
```

SheduledExecutorService

Необходим тогда, когда необходимо расписание на запуск потоков

Интерфейсы Callable и Future

Callable позволяет возвращать значения и выбрасывать исключения

Semaphore

Semaphore – это синхронизатор, позволяющий ограничить доступ к какому-то ресурсу. В конструктор Semaphore нужно передавать количество потоков, которым Semaphore будет разрешать одновременно использовать этот ресурс.

acquire

release

CountDownLatch – замок с обратным отсчетом операций

Отсчитывает необходимое количество операций и после этого запускает потоки

Exchanger

Нужен для обмена информацией между двумя потоками.

AtomicInteger

AtomicInteger – это класс, который предоставляет возможность работать с целочисленным значением `int`, используя атомарные операции.

`incrementAndGet`

`getAndIncrement`

`addAndGet`

`getAndAdd`

`decrementAndGet`

`getAndDecrement`

Синхронизированные коллекции

Коллекции для работы с многопоточностью

Synchronized collections

Получаются из традиционных коллекций благодаря их обёртыванию

Concurrent collections

Изначально созданы для работы с многопоточностью

`Collections.synchronizedXYZ(коллекция)`

ConcurrentHashMap

CopyOnWriteArrayList

Небольшое количество операций по изменению

Большое количество операций по чтению данных

Раздел 09 Работа с файлами IO и NIO

FileReader & FileWriter

FileReader и FileWriter используются для работы с текстовыми файлами.

```
FileWriter writer = new FileWriter("file1.txt");
```

```
FileReader reader= new FileReader("file1.txt");
```

Никогда не забывайте закрывать стримы после использования.