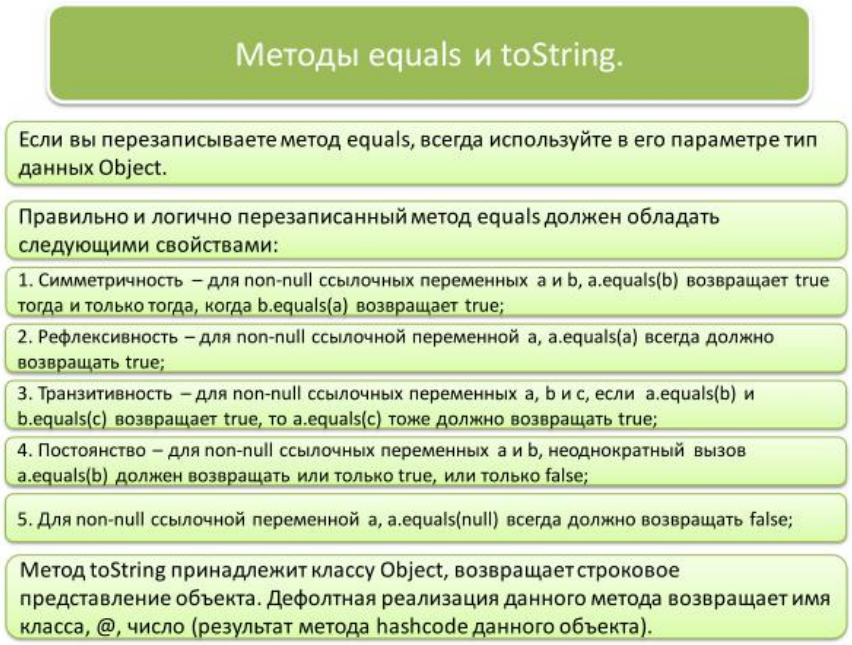
Equals

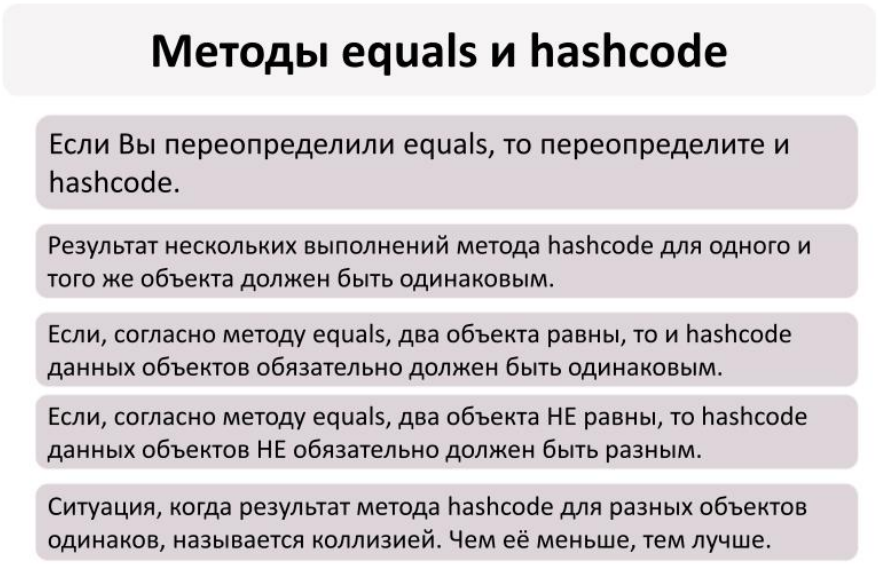
По умолчанию equals **идентичен** “==”:

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



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.



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)
```

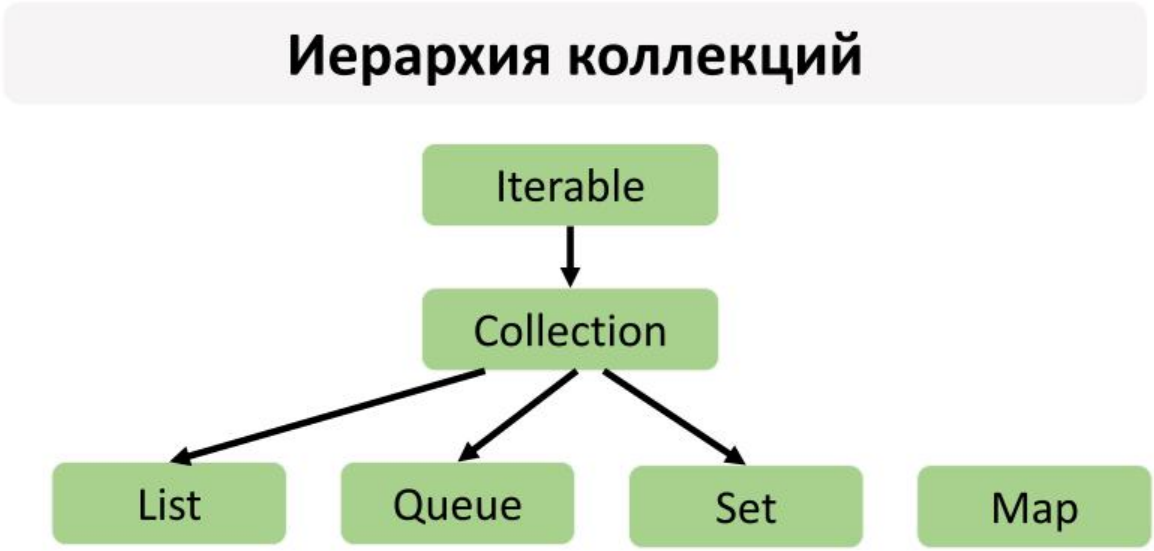
Wild card

Нельзя добавить элемент в список с неизвестным типом элементов

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

Нужно использовать bounded wild card

Collection



LinkedList<>
Связанный список
Dovbly 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 hascode 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 отличии 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.

Daemon 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

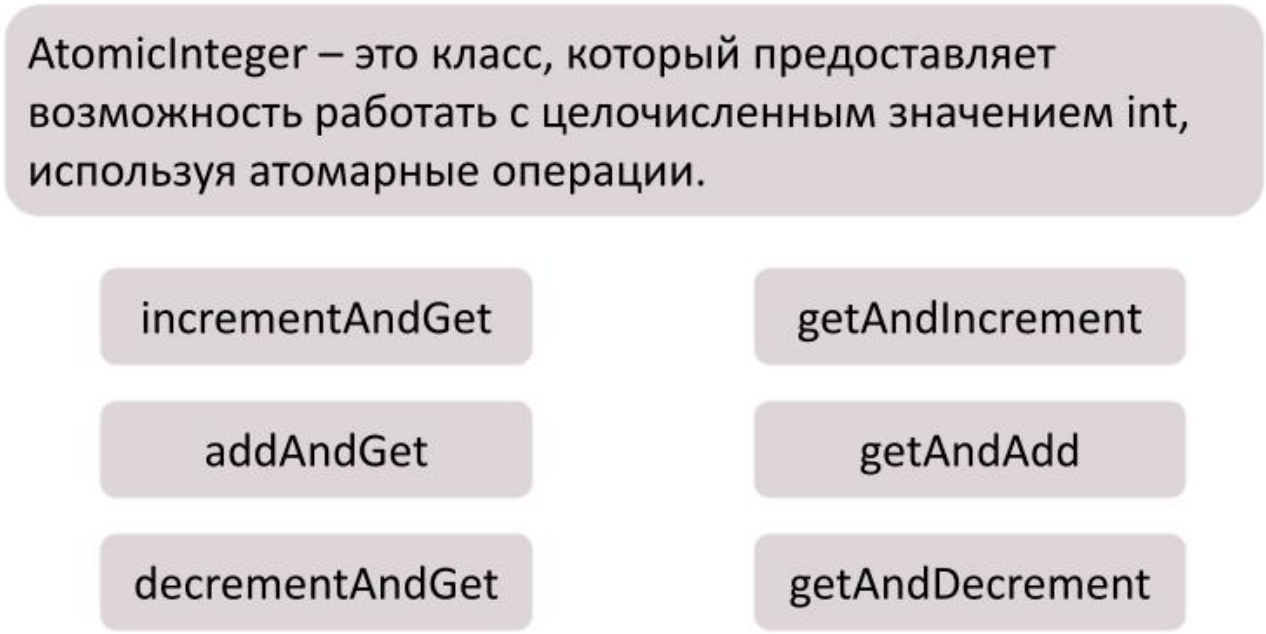
Интерфейсы Callable и Future
Callable позволяет возвращать значения и выбрасывать исключения

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.

CountDownLatch – замок с обратным отсчетом операций
Отсчитывает необходимое количество операций и после этого запускает потоки

Exchanger
Нужен для обмена информацией между двумя потоками.
AtomicInteger



Синхронизированные коллекции

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.