JAVA Cheat Sheet 3

Basic OO

Reference types

In JAVA only primitive types are handled **by value**, because their representation does have a fixed size.
All other objects are handled **by reference**.
Thus **only** their **references are passed**, not their values → Operations on variables storing references will affect other variables holding the same reference.

Objects, classes, instances

Every JAVA program is a set of **classes**.
Every piece of code has to be part of a class.
A class is the **concept** or **blueprint** of an object.
Classes can be **instantiated**, creating a concrete object from those blueprints - an **instance**.
Every class is inherited of the class **Object**.

Attributes (fields, members)

A class can have several **attributes** describing it. Those attributes are typically declared at the top of the class and their scope is the whole class. They are called **fields** or **members**.

Advanced OO

Modifiers (class, attribute, method)

Visibility

- **public** → Accessible from everywhere
- **private** → Accessible only from within this class
- **protected** → Accessible from this class, all sub classes and all classes in the same package.
- package private (no keyword) → Accessible from this class and all classes in the same package

static

Can be accessed without having an instance of the class.
Holds for every instance.

final

Final **classes** may **not be subclassed**.
Final **methods cannot be overridden** by subclasses.
Final **attributes cannot change their values** once initialized.

Constructors, initializers, memory allocation

Constructors

Methods called when **initializing** an Object.
Do not specify a return value.
Name has to equal the class' name.
Every object provides a **default constructor**.
Usually used to initialize fields with user given values.

Initializers

Code that **does not belong to any method** inside of a class.
Declaring (and initializing) fields is a common usage.
Will be executed in order of appearance. [1]

Memory Allocation

JAVA objects reside in the **heap**.
The *heap* is divided into **young space** and **old space**.
The **JVM** distinguishes between **small** and **large objects**.
Small objects get allocated in **thread local areas (TLAs)** of the *heap*.
Large objects are allocated in the *old space*.[2]

Destructors, garbage collection

Destructors

In JAVA there are no destructors. Destroying objects is **done by the garbage collector**, but there is no exact telling when this will be done.

Garbage Collection

[3] **Garbage collection** is the process of freeing space in the *young space* for the allocation of new objects.
The *JVM* uses **mark and sweep** for garbage collection on the **whole heap**.
It **marks** all objects as *alive* that are reachable from threads, native handles and other root sources and also those objects reachable from *alive* objects. The rest is considered garbage.
It then identifies the free space between the *alive* objects. This is called **sweeping**.
Garbage collection on the **young space** is called **young collection**.
The **young collector** promotes all live objects outside a so called **keep area** to the **old space**.
A garbage collection strategy using a *young space* is called **generational garbage collection strategy**.
There are several **garbage collection modes**:
dynamic modes:

- throughput → maximum application throughput
- pausetime → short and even pause times
- determinicstic → very short and deterministic pause times

static modes:

- singlepar → single-generational parallel garbage collector
- genpar → two-generational parallel garbage collector
- singlecon → single-generational mostly concurrent garbage collector
- gencon → two-generational mostly concurrent garbage collector

Compaction is used to **get rid of small chunks of space** in the heap and to instead **create bigger chunks** by moving objects in the heap closer together.

Inheritance

In JAVA except for the class *Object* every class has exactly one **superclass**.
A **subclass** inherits fields and methods from its superclass.
Eventually, every class inherits from the class *Object*.
To subclass another class use keyword **extends**.
public class Audi ***extends*** *Car{...}*

Polymorphism

In JAVA every class is also an instance of its superclass.
As for this holds also for the superclass, every class in JAVA may be an instance of many different classes.
Eventually every class is an instance of *Object*.
possible:
Car car = new Audi(); ← if Audi extends Car.
But now only functionality of type Car is accessible, even if the assigned type is an Audi.

Local, inner, anonymous classes

An **inner class** is a class defined within another class. So the inner class can only be accessed from an instance of the outer class.
A **locale class** is a class that is defined in a block of another class, typically a method.
A local class can only be used within the block where it is declared - it has **local scope**.
OuterClass.InnerClass innerObject = outerObject.new InnerClass(); [4]
An **anonymous class** is a class that is declared and instantiated at the same time.
Car car = new Audi(){...<class definition here>...}

Modularization, packages, archives

Packages and **archives** are ways to **modularize code**.
Packages have a hierarchical **tree-like structure** and function as folders for code.
JAVA archive files (JARs) are archives containing packages and their contained classes.

---

[1] see http://www.dummies.com/how-to/content/what-is-an-initializer-in-java.html
[2] see https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html
[3] see https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html
[4] see https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html