**Inner & Nested Classes**

– COMMON to all: they can extend ANY class

– COMMON to all non-static: they may specify static members only if these are compile-time constants

– COMMON to all static and non-static:

– being just a member of Outer doesn't make Inner its child without extends 🡪 this.some\_member\_in\_Outer inside Inner won't work:

class Outer

{

int i = 42; // if this var were marked private, #1 wouldn't have compiled but #2 still would

class Inner

extends Outer // if commented out makes #1 fail compilation

{

public void run()

{

System.out.println(i); // Outer's members are visible to Inner

System.out.println(this.i); // #1

System.out.println(Outer.this.i); // #2 accessing i thru its full name and not thru inheritance

}

}

}

– **member inner classes (incas)**

– cannot declare anything static (fields, methods or initializers) except compile-time constants (that is, nothing created with new, computed at run time + Wrappers)

– there is ALWAYS an outer class's object associated with an inca's instance (explicit or, most commonly, implicit this 🡪 this.new Inca() or just new Inca() inside a static method throws a comperr); incas need an enclosing class's object to instantiate themselves:

class Test{

class Inner{}

static class Instat{}

void run(){

Inner inner3 = this.new Inner();

}

public static void main(String[] args) {

Inner inner1;

Test.Inner inner2; // another syntax for declaration

Test test = new Test();

inner1 = test.new Inner();

inner2 = new Test().new Inner(); // another syntax for instantiation

// Inner inner3 = new Inner(); // INVALID since new Inner() is equiv to this.new Inner() and **this** is not static

// Inner inner3 = Test.Inner(); // INVALID since Test has no static method Inner()

//--- and now to the static nested class... ------------

Test.Instat instat1 = new Test.Instat();

Instat instat2 = new Test.Instat();

Instat instat3 = new Instat();

}

}

class Test2{

// Inner inner; // INVALID: can't find symbol 'class Inner'

Test.Inner inner = new Test().new Inner();

Test.Instat instat = new Test.Instat();

public static void main(String[] args) {

Test.Inner inner = new Test().new Inner();

Test.Instat instat = new Test.Instat();

}

}

– **local inner, a.k.a. method local classes (melocas)**

– even inside static methods they can't declare static members + static initers + but static compile-time constants are OK (see clarifications above)

– can be used inside methods and code blocks such as initializers, conditional constructs and loops

– can access only final or effectively final local vars of the enclosing method; clarification: till 1.7 such vars needed to be marked as final explicitly.

– name of the method local class is outer class name + $ + integer + inner class name: Outer$1Inner

– **anonymous inner classes (anoncas)**

– can extend a class or implement an interface – but not both!

– must implement abstract methods declared in the superclass or superinterface

– can't have constructors

– can have initing params (passed to super's ctor) when extending a class

– vars and methods of the anoncas can't be accessed from outside the anon 🡪 be on alert when an anonca declares something (most likely, a method) while the anon's parent doesn't have such a method 🡪 comperr if the method is called from outside the anon

– anoncas created inside static methods can't see instance vars; it's because such anoncas have no reference to the object that holds the instance var:

class One{}

class Test{

String str = "";

<T>void run(T t){}

public static void main(String args[]) {

Test test = new Test();

test.run(new Test() {

void run(){

System.out.println(str); // VALID

}

} );

test.run(new One() {

void run(){

System.out.println(str); // INVALID: non-static str in static context

}

} );

}

}

– **static nested classes (stincas)**

– you call **new Outer.Inner()** rather than Outer.new Inner() because Outer.Inner is the static class's name 🡪 dot opens only the list of Outer's static members + class keyword.

– a static nested class can contain non-static members including non-static inner classes because declaring a nested class static only means that instances of the class are created without having an outer instance.

– "A static nested class interacts with the instance members of its outer class (and other classes) just like any other top-level class. In effect, a static nested class is behaviorally a top-level class that has been nested in another top-level class for packaging convenience." (from <http://docs.oracle.com/javase/tutorial/java/javaOO/nested.html>)

– when using import static remember that it will import the stinca only 🡪 Inner inner = new Inner() is OK but Outer.Inner won't be visible 🡪 add one more import stat such as import otherpack.Outer, etc.