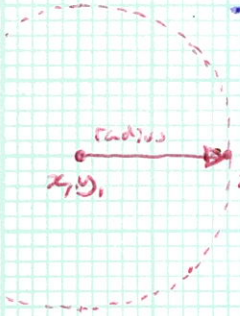
*SENG2200 - Homework Challange II - Code Stuff

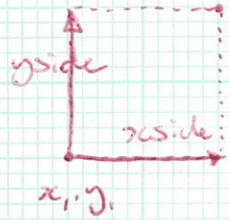
Pt1, create & test the following structure in Java:

a) Abstract Shape class - write an <<abstract>> class called 'Shape', with i) protected members x1, y1, x2, y2 and area of type double, & ii) public abstract method called getArea() that returns a double.

b) Concrete Circle & Quad class - extend your abstract 'Shape' class with TWO concrete classes: Circle & Quad

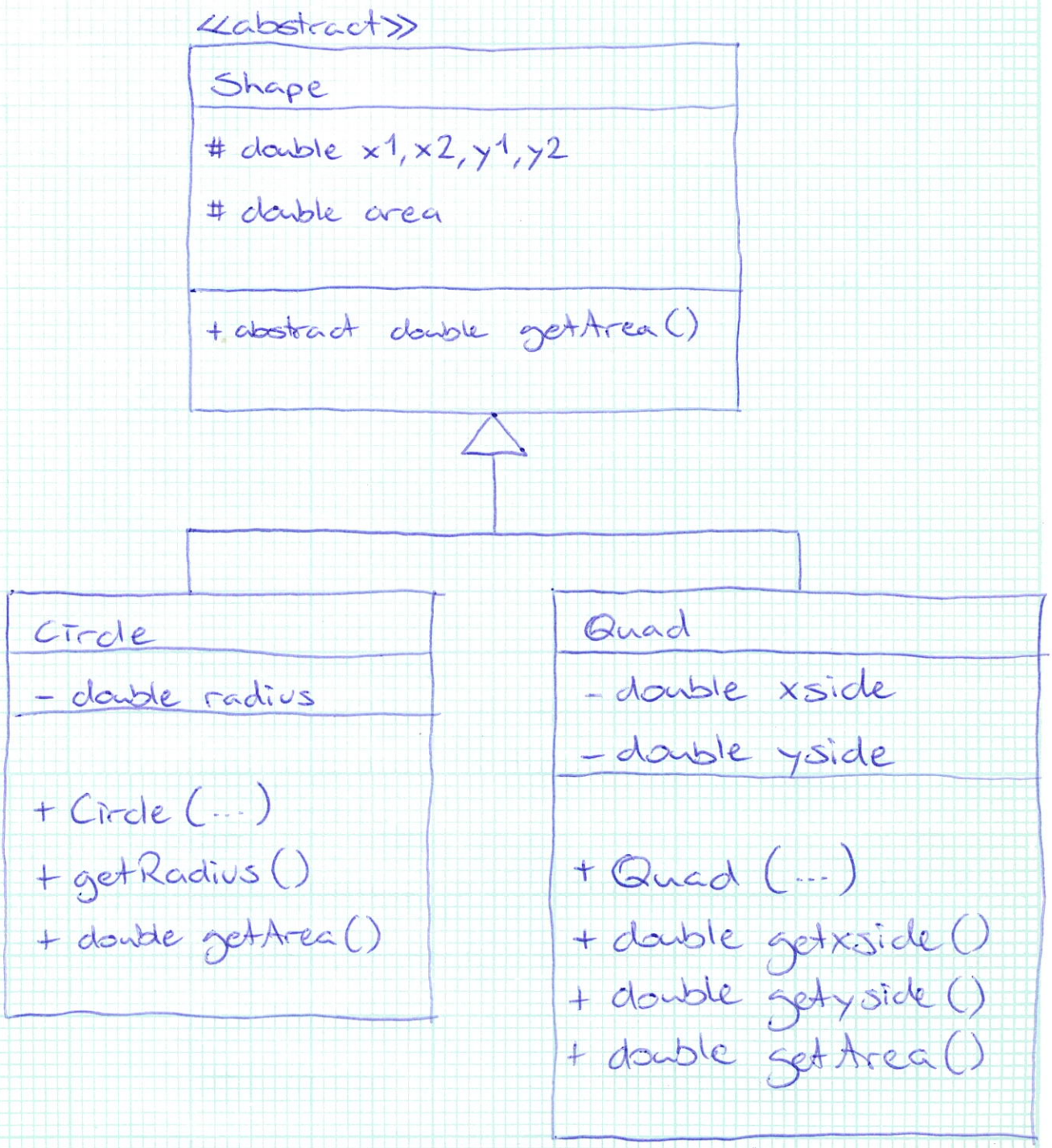


Circle will have i) member of type double called radius, ii) methods (public) of Circle(double x1, double y1, ...etc...), double getRadius(), and iii) an implemation for getArea()



Quad will have i) members of type double called xside & yside, ii) public methods for getxside(), getyside() & Quad(double x1, double y1, ... etc...), and iii) an implementation for getArea()

pt1.
1.



c) Write some simple code to test this stuff.

* Stuff to think about
  * Using Circle & Quad "polymorphically" as "Shape"
  * type checking with the "instanceof" operator.

pt1.

2.

*SENG2200 - Homework Challenge II: Abstract/Interface: WTF?

PT2/

OK, Firstly: Java Abstract & Java Interfaces - I've noticed some resources say that 'interface' in Java is obsolete, but the current JDK & API still list both as valid; so just to cove the differences:

① an Abstract can extend ONLY one class or Abstract at a time - an Interface can extend any numbe of interfaces.

② an Abstract class can be inherited by a Class or an Abstract - Interfaces can only be extended by Indefaces & require Classes to implement them (as opposed to extend).

③ an Abstract can contain both Abstract & Concrete methods - an Interface contains only Abstract methods; they can not have Concrete methods.

④ a Class can extend only one Abstract at a time - but a Class can implement any numbe of Interfaces at a time

⑤ in an Abstract Class, the 'abstract' keyword is mandatory for a method to be abstract - an Interface all methods are abstract, so the keyword is optional. →

pt2.
1.

⑥ Abstract classes have 'protected', 'public', & 'public abstract' methods - Interface methods are 'public' by default.

⑦ Abstract can have 'static', 'final', & 'static final' variables, with 'private', 'protected', & 'public' access modifiers - an Interface (by default) has only 'static final' (ie 'constants').

Yep - that's pretty heavy, yeah? So according to Oracle & Java 'Best Practices' in General:

*Abstract:
- when you want to share code amongst closely related classes;
- when you expect the classes you use to extend an 'abstract' to have many common members & methods, or require access modifiers other than 'public'; and
- when you want to declare non-constant members.

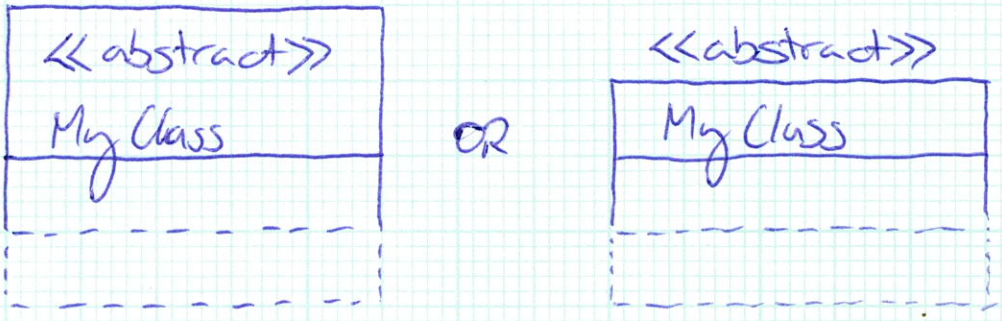
eg: Abstract Map.

*Interface:
- when you expect unrelated classes to implement the 'interface' - eg: Comparable.
- when you want to specify behaviour of a data type, but are not concerned about who implements this.
- when you want to take advantage of 'Multiple Inheritance' of type. eg: Hash Map.

* UML & Design with Abstract & Interface

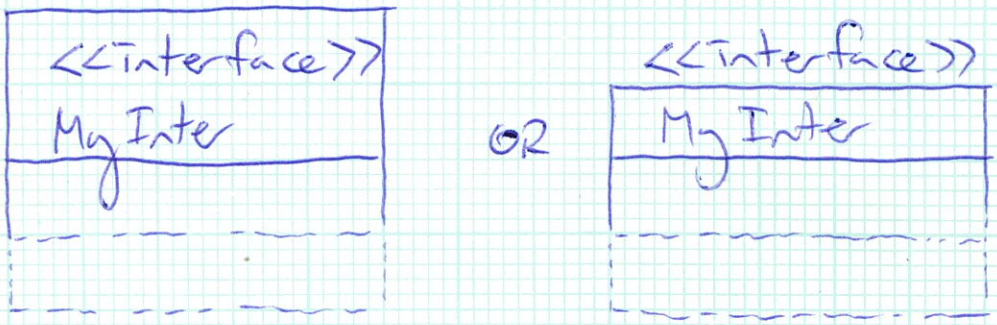
Usually the convention is as follows:

* Abstract (because you will mostly write these):



* NOTE: you may also see the 'abstract' identifier in braces: eg {abstract}

* Interfaces (that you write yourself):



* Interfaces (that come from the Library) - can be done with 'lollipops':