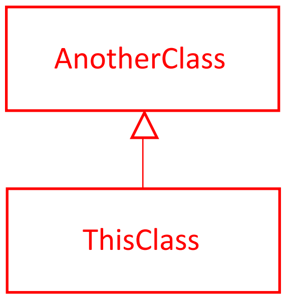
# **EECS1720**
# **Worksheet 3 – Inheritance**

1) How do you determine whether a class extends another given its API? What if you were given the UML class diagram?

API → keyword extends used in the signature of the class..
e.g. `public class ThisClass extends AnotherClass { … }`

UML →



2) Given that a class extends another, what can you say about the features of the two classes?

ThisClass (the extending class) inherits all the features of AnotherClass (the extended class) – if considering the example in (1).

I.e. ThisClass inherits (incorporates) all the fields and methods from AnotherClass, plus has the ability to include any new fields/methods defined in ThisClass.

Private fields and methods are part of any ThisClass object, however are not directly accessible through ThisClass and must be invoked/accessed through the public api of the parent class (AnotherClass) – if available; while public/protected fields and methods of AnotherClass <u>are</u> directly accessible from ThisClass.

3) If a class extends another, why is it thought of as a specialization?

Because it has all the properties of the class being extended (so it is considered "is-a" version of the class being extended), plus it has additional defining fields/methods, so its state and behaviour are even more specific than the class being inherited from.

The parent class defines more generic components in common between the two classes, while the child builds upon this. The child thus has properties the parent does not (more specialized), while the parent does not have any properties that the child does not also have.

Another way to think about it. The child can do everything the parent can do… plus more

4) Argue that an inheritance hierarchy can be thought of as several inheritance chains with a class in common.

   Any class can only have at most one parent.. so two given classes can possibly share a common parent. If this occurs, a hierarchy forms (class tree or several trees).

   In java, since all classes inherit from Object (unless otherwise specified), the root of such hierarchies is always Object.

5) Is it possible for a class to have two children? How about two parents?

   A class may have two children, A class may only have one parent (single inheritance)

6) What is “single inheritance”?

   The restriction that a class may only inherit (extend) from a single class

7) What is “multiple inheritance”?

   When a class may inherit (extend) from multiple classes – supported in some languages (e.g. C++), but not supported in java

8) If a subclass has a method with the same name as an existing parent’s method, which method will appear in the subclass API? Will your answer change if the child’s method has a different parameter list than that of the parent?

   Depends. If both methods have the same signature, then the child version overrides the parent version of the method. If both methods have different signatures, then the child has both in its API (assuming the parents method is either protected or public). If the parents method is private, then it is not available in the API of the child.

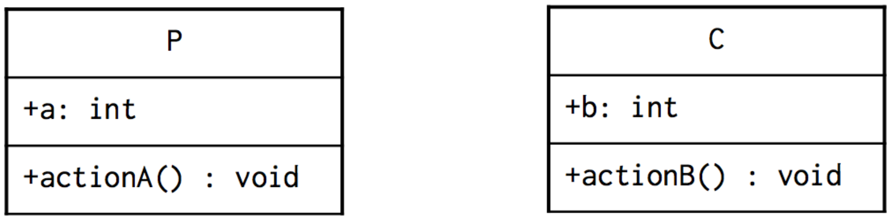
9) If a subclass has a field with the same name as a parent’s field, which of the two will appear in the subclass API? Will your answer change if the child’s field has the same type as that of the parent?

   The subclass field will shadow the parent’s field (if same name), i.e. you wont be able to access the parents version of the field from the child (regardless of type).

10) Given UML class diagrams for C and P, determine fields and methods that would be found in C's API (like the following figure)

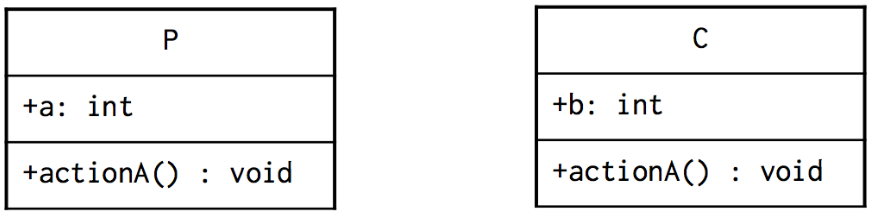
Lets assume C extends P, show the API for C given each of the following =>

(i)



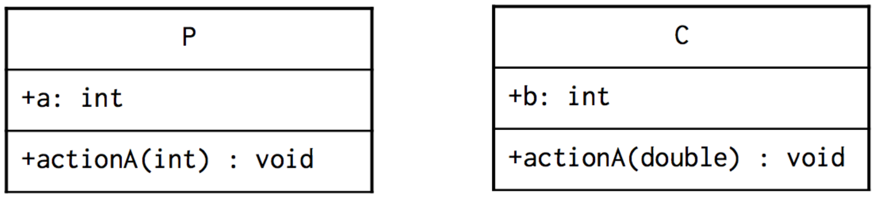
Fields: a, b Methods: actionA(), actionB()

(ii)



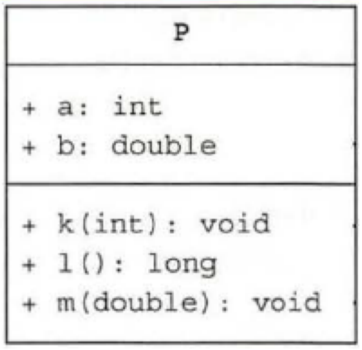
Fields: a, b Methods: actionA() from C only (overrides P)

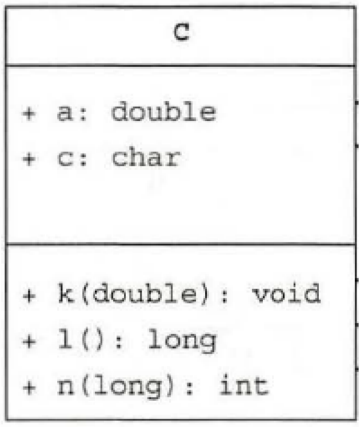
(iii)



Fields: a, b Methods: actionA(int) from P, actionA(double) from C (both available as they have diff signatures)

(iv)





Fields: a:double, b, c
(a in C shadows a from P)

Methods: k(int):void, k(double):void, l():long from C (overrides P's version), m(double):void and n(long):int

11) What is an interface?

A declaration of a collection of methods (and fields). i.e. the method signatures are provided only. A class that implements the interface is forced to define (implement) all of its methods

12) How are the following "features" represented in UML?

a. Aggregation
(line connecting classes with open diamond connected to class that holds a field of the other class)

b. Composition
Same as above except a closed diamond (not open one)

c. A standalone class
Box with three sections (top = class identifier, middle = fields & types, bottom = constructor and method signatures and method return types)
(see lecture slides)

d. A class with private methods and public attributes?
Fields have '+' in front of them, methods have '-' in front of them

e. A static field?
Field name is underlined

f. A static method?
   Method name is underlined

g. An interface?
   Title has “ <<interface>> “ above it

h. Private vs. public vs. protected fields / methods?
   Private has ‘-‘, public has ‘+’, protected has ‘#’ in front of the field or method name

i. Inheritance
   A solid arrow connects the child to the parent class (arrow points to parent)

j. Interfaces
   Similar to inheritance, however arrow has a broken line, arrow connects a class that implements the interface to the interface

13) What is a protected field and when is it useful?

A protected field is a field that acts like a <u>private</u> field (no external access to the field) except for any classes extending the class in which it is defined.

For subclasses only, a protected field acts like a <u>public</u> field. It is useful if you trust a subclass to have direct access to a field/method, so can simplify a subclasses ability to mutate fields inherited from the parent without having to implement a large number of accessor/mutator methods in the parent

14) Which of the following can be instantiated?

a. An Interface

**b. A Regular (concrete) Class**

**c. A Base Class that extends from a Regular class**

15) Are there situation(s) in which a class may not be instantiated? Explain.

If a private constructor is defined (and it is the only constructor) – this prevents any external class or subclass from calling the constructor (thus creating/initializing an object).

If a private constructor is not defined, and there are no other constructors, then a public default constructor will be “implicitly” included by java at compile time, so may be instantiated. Utility classes (like Math – i.e. all static features) should therefore include a private constructor to prevent instantiation.

*** note an interface is NOT a class. It also cannot be instantiated*

16) What is meant by the following?

a. Overloading

many methods with same name (but diff signatures)

b. Overriding

when a method in a subclass has same signature as parent method (but diff implementation)

c. Shadowing

when a field in a subclass has same name as a field in parent, any reference to that field in the child will be unable to access the parent version of that field

17) What is the purpose of super( … )  ??

super(…) allows a constructor of a child class, to invoke one of the constructors of the parent class directly (again only has the first statement of a constructor in the child class, and never from anywhere else other than a constructor in the child class).  The signature of this super call (its arguments) determine which constructor is invoked in the parent.

18) What is the "substitutability" principle?

The idea that any object instantiated from a child class, may be assigned to a reference of any of its ancestors (parent class or parents of parents up the inheritance chain).

This conforms to the notion that a child "is-a" version of its parent, which "is-a" version of its parent, etc…