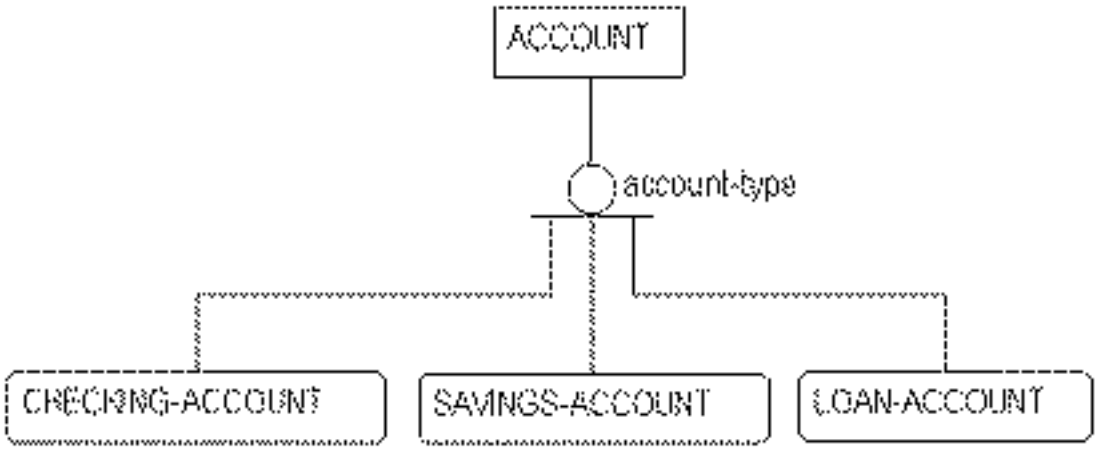
Subtyping

Matthew Hambrecht
Department of Computer Science - UMBC

created for
CMSC 331: Programming Languages
Prof. Lupoli
05/08/2023

Subtyping is a key part of object-oriented programming due to its ties to polymorphism and inheritance. The inclusion of subtyping allows programmers to connect classes and datatypes that contain similarities in member variables/functions without having to make a copy of the similar code. Take for example four classes: Account, CheckingAccount, SavingsAccount, LoanAccount.



In this example the supertype would be “Account”. The “Account” class could store member variables such as “balance”, “creation date”, “owner”, etc. These are generic pieces of data that could also be used my more specialized types of accounts. These specific accounts may need extra member variables and functions to achieve their intended purpose but if they share the same signature as the member methods and variables of the supertype “Account” they are considered subtypes of it. This concept seems like inheritance as they are often coinciding in object-oriented programming, but they are two separate concepts. This difference can be seen between a comparison between Int32 and Int64 datatypes. Int32 and Int64 are unrelated by inheritance, however, all pieces of 32-bit integer data can be stored as 64-bit integer data. Thus, Int64 is a subtype of Int32. A more detailed example of the difference between inheritance and subtyping by computer science researcher Madhavan Mukund at Chennai Mathematical Institute is:

> “Consider the data structure deque, a double-ended queue. A deque supports insertion and deletion at both ends, so it has four functions insert-front, delete-front, insert-rear and delete-rear. If we use just insert-rear and delete-front we get a normal queue. On the other hand, if we use just insert-front and delete-front,

we get a stack. In other words, we can implement queues and stacks in terms of deques, so as datatypes, Stack and Queue inherit from Deque. On the other hand, neither Stack not Queue are subtypes of Deque since they do not support all the functions provided by Deque. In fact, in this case, Deque is a subtype of both Stack and Queue!”

In programming having the ability to subtype allows for more modular code design, increased code reusability, and decreased code size. These concepts allow for developers to create cleaner and safer code. This can be seen through the example of the double-ended queue. Say we’re making our own implementation and using an ArrayList as our storage type. We could create a Stack class with FIFO functionality for an array and a Queue class with FILO. Since we know that a Deque is a subtype of a combination of a Stack and a Queue since it uses the same operation signatures, we are given the option to create a third data type with minimal additions by extending both classes into a subtype of Deque. This creates a multi-purpose data-structure that can be used in lieu of either as it will always support the same methods that the supertype has. It also allows us to more easily trace back issues that may result from certain functionality as we know exactly where the issue lies depending on which superclass the broken functionality is shared with. Another useful aspect of subtyping is type-safety. Type-safety is a feature that ensures a program doesn’t perform invalid operations on certain datatypes. The reason subtyping increases type safety is that if we know an Object is a Subtype of another object, we don’t have to assume that certain methods are implemented because the requirements to be a subtype assume that it is always safe to use any Subtype in any context where their Supertype is required (but not vice-versa due to extended functionality). Hence why something like this is permissible:

```
1  public void someMethod(Number n){
2      // some actions
3  };
4
5  someMethod(new Integer(10));
6  someMethod(new Double(10.1));
```

We care about subtyping for the exact same reason that it exists: it benefits our code structure through its ability to promote modularity, type-safety, and reusability. Java has interfaces which are the key to subtyping. Interfaces differ from standard inheritance in that they don't provide underlying information about implementation, rather they provide the signatures expected of methods being derived providing an abstract template for all its subtypes. An example of this using a Vehicle supertype with Plane and Car subtypes can be seen here:

```
1  interface Vehicle {
2      void whoAmI(String name);
3  };
4
5  class Car implements Vehicle {
6      public void whoAmI(String name) {
7          System.out.println("I'm a " + name + " car");
8      }
9
10     public void driveForward() {
11         // cool stuff
12     }
13 };
14
15 class Plane implements Vehicle {
16     public void whoAmI(String name) {
17         System.out.println("I'm a + " + name + " plane");
18     }
19
20     public void takeOff() {
21         // cool stuff
22     }
23 };
24
```

For something to be a subtype "the methods of the first type must have signatures that **conform** to the signatures of those in the supertype." In this example both subtypes use the same signature of the whoAmI function as the parent, whilst also adding their own implementations. However, this example shows why understanding the difference between inheritance and subtyping is important as the whoAmI implementations are similar enough in the subtypes that it would be much cleaner to turn Vehicle into a parent class and implement it there. Hence, understanding what subtyping is and when

to use it over inheritance (compatibility vs code reusability) is key to being a good developer.

Sources

baeldung. (2022, July 23). *Type safety in programming languages*. Baeldung on Computer Science. https://www.baeldung.com/cs/type-safety-programming

Department of Computer Science. (n.d.). https://www.cs.utexas.edu/~wcook/papers/InheritanceSubtyping90/CookPOPL90.pdf

Interfaces and subtyping. Home. (n.d.-a). https://www.cs.cornell.edu/courses/cs2112/2016fa/lectures/lec_subtyping/

MIT OpenCourseWare. (n.d.). *Laboratory in software engineering: Electrical engineering and computer science*. MIT OpenCourseWare. https://ocw.mit.edu/courses/6-170-laboratory-in-software-engineering-fall-2005/

Subtype polymorphism. Home. (n.d.-b). https://www.cs.cornell.edu/courses/cs4120/2022sp/notes/subtyping/

Subtype Relationships. Data Modeling Overview Guide release 9.5.0. (n.d.). https://s3.amazonaws.com/erwin-us/Support/95/CA+ERwin+Data+Modeler+r9+5-ENU/Bookshelf_Files/HTML/ERwin%20Overview/index.htm?toc.htm%3F254734.html

The Trustees of Princeton University. (n.d.). *Programming languages: Subtyping*. Princeton University. https://www.cs.princeton.edu/courses/archive/spr96/cs441/notes/l21.html