Inheritance

What is it?

Abstractions - we do it all the time

Let’s say we were to write a file manager (or file browser). What types of objects is it it should handle?

Files, right?

It should be able to display a list of files in some directory and perhaps also to display each file as a thumbnail representation.

What type of files should it be able to list and display thumbnails for?

Does it matter? A file is a file, right?

File is an abstraction

It would be nice if we could treat all types of files the same way, regardless of whether it is a text file, audio file, video file, or some other type of file.

It should list the name and some kind of thumbnail representation.

So we could use File as an abstraction to mean “Any file which has a name and is capable of giving us its name and rendering some kind of thumbnail representation of itself” in our system.

A File could be a type, but are there many kinds?

So, our file manager could deal with File objects, that is, we could treat every actual file on the file system as a File object. File would then be an abstraction and we could talk about File as a type of objects in our system.

But there are many kinds of files. The thumbnail of an image should probably be different from the thumbnail of a video file etc.

We could think of AudioFile as a subtype of File. But we know that there are actually different types of audio files as well, so subtypes can have subtypes!

All files, however, share some common characteristics and behavior.

Another example

Let’s say we also were to write some kind of media player. Our media player should be able to find and read File objects from the filesystem. But we would be particularly interested to read media files, let’s say for instance audio files and video files.

MediaFile could be a subtype of File, and it could in turn have the subtypes AudioFile and VideoFile.

All media files could share the behavior of being playable, that is, invoking the method play() would make the files present their media (audio or video+audio).

What do we gain by all this nonsense?

One thing that we gain from using abstractions and using types and subtypes is that our application for playing media files can focus on what a MediaFile is in general (on an abstract level) which would make it capable of doing a few things which are common to all MediaFile objects (as we have defined them), but care less about what particular subtype of MediaFile it is dealing with at the moment.

As long as both the MediaFile subtypes AudioFile and VideoFile have a behaviour present() described in the supertype MediaFile, the player application can deal with any type of MediaFile and just present it.

Inheritance as a means of expressing subtypes

In Java, we can create subtypes using something called inheritance.

We can create hierarchies of types using some kind of inheritance mechanism.

For now, you can think of it as defining all common treats in some kind of supertype (like File or MediaFile) and specifying how these treats are implemented differently in the subtypes.

Using supertypes and subtypes

If we have a reference to a File (as we have defined it in some class) we can be sure that we can call certain methods on it like thumbnail() or name() etc because these methods are declared in our File class.

And if we have a reference to a MediaFile object, we are certain that we can call present() on the object, since this method is declared in the MediaFile class (which we have written).

A file manager can call thumbnail() on all objects which are of a subtype to File, because subtypes keep the API of the super type, but might do it differently. We can call thumbnail, but the result may vary!