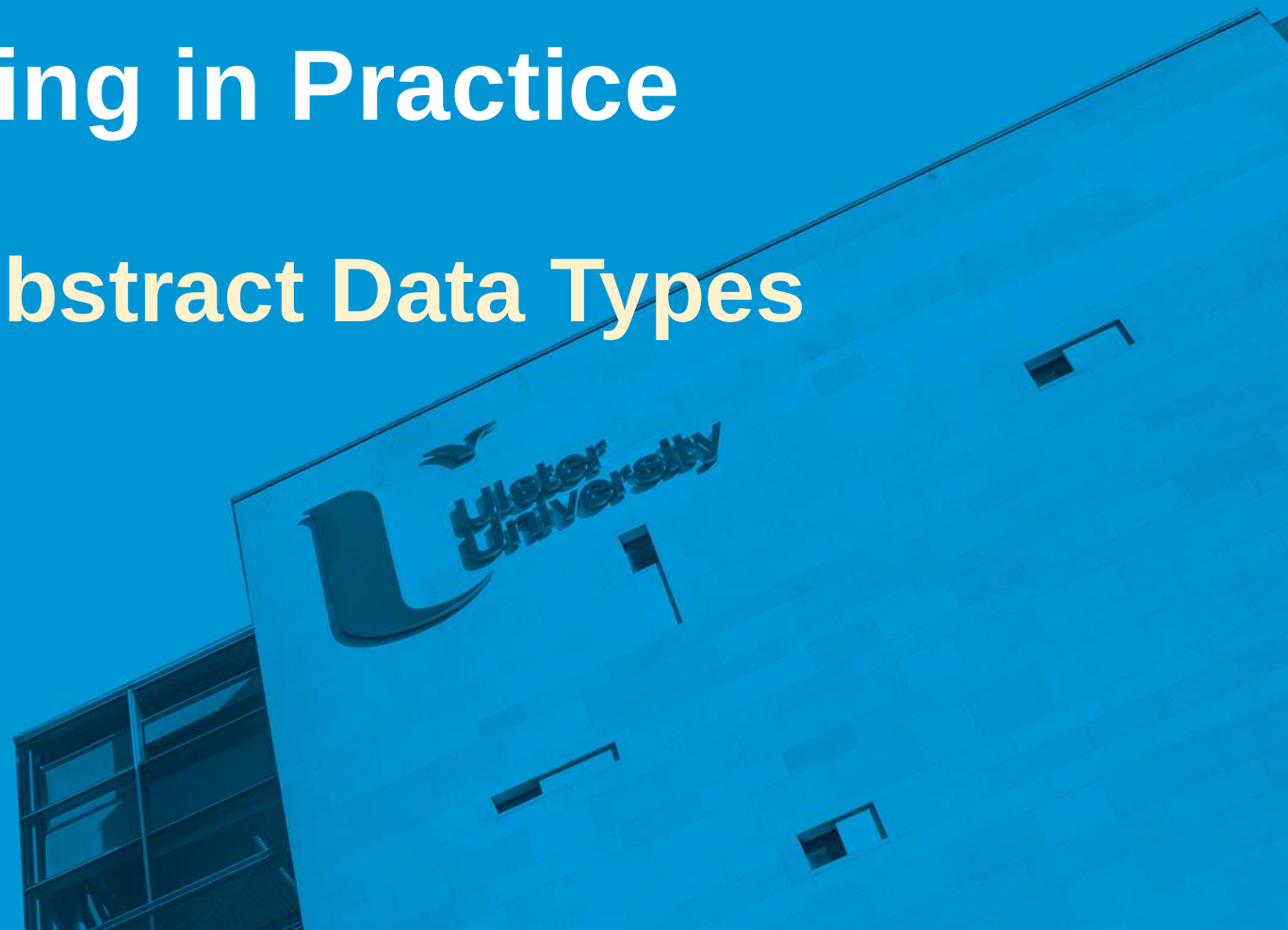
Ulster University

COM410 Programming in Practice

A3.1 Introduction to Abstract Data Types



Abstract Data Types

- A data type is set of values and operations on those values (defined within a specific programming language)

- Primitive types:
  - values map to machine representations
  - operations map to machine instructions

- We want to write programs that process other types of data!

type	set of values	examples of values	examples of operations
char	characters	'A' '@'	compare
String	sequences of characters	"Hello World"	concatenate
int	integers	17 12345	add, subtract, multiply, divide
double	floating-point numbers	3.1415 6.022e23	add, subtract, multiply, divide
boolean	truth values	true false	and, or, not

- An abstract data type is a data type whose representation is hidden from the client

Abstract Data Types

- A more specific definition:

An **Abstract Data Type** (ADT) is a specification for a group of values and the operations on those values that is defined conceptually and independently of any programming language.

(a data structure is an implementation of an ADT within a programming language)

Abstract Data Types

- You have already seen the implementation of an ADT when manipulating string values

- Java's `String` data type is an ADT that allows programs to manipulate strings
- A String is a sequence of Unicode characters
- The API for String provides an interface for the ADT
- We can use an ADT without knowing the underlying implementation details

public class String	
String(String s)	create a string with the same value
int length()	string length
char charAt(int i)	ith character
String substring(int i, int j)	ith through (j-1)st characters
boolean contains(String sub)	does string contain sub?
boolean startsWith(String pre)	does string start with pre?
boolean endsWith(String post)	does string end with post?
int indexOf(String p)	index of 1st occurrence of p
int indexOf(String p, int i)	index of 1st occurrence of p after i
String concat(String t)	this string with t appended
int compareTo(String t)	string comparison
String replaceAll(String a,String b)	result of changing as to bs
String[] split(String delim)	strings between occurrences of delim
boolean equals(String t)	is this string's value the same as t's

- Other ADTs we have already designed and used include `Building`, `House`, `Shop`, `Pet`, `Cat`, `Dog`, etc.

Abstract Data Types

- When using data types, you need to know:
  - Its name (capitalized in Java)
  - How to construct new objects (instantiation)
  - How to apply operations on to a given object (invoke methods)

- To construct a new object:
  - Use keyword **`new`** to invoke a constructor
  - Use data type name (class name) to specify the type of the object

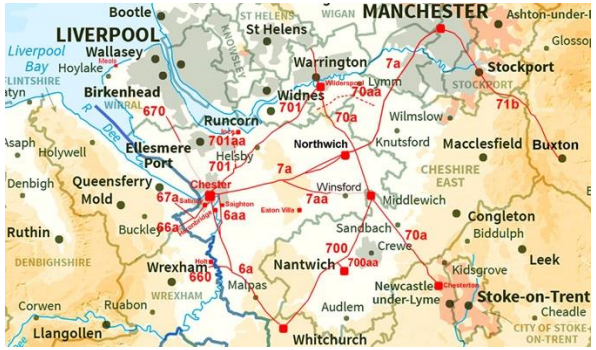
- To apply an operation:
  - Use object name to specify which object
  - Use the dot operator to indicate an operation is to be applied

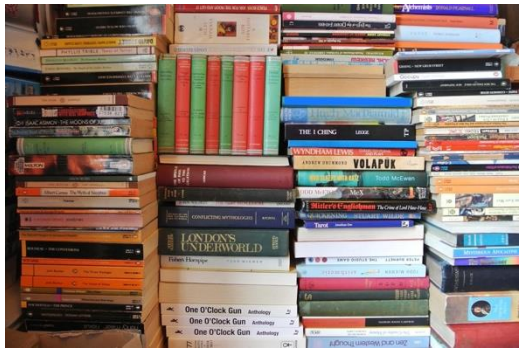
```
Building b1;
b1 = new Building("101 Main Street",
                  "Homer");
b1.setOwner("Marge");
System.out.println(b1);
```

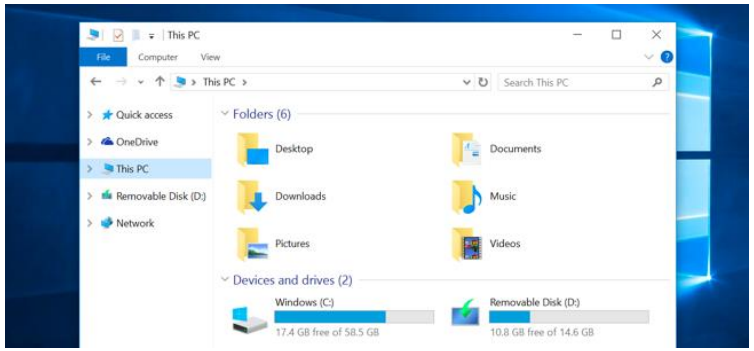
Using an ADT is like using a vending machine

- Can perform only tasks machine's interface presents
- You must understand these tasks
- Cannot access the inside of the machine
- You can use the machine even though you do not know what happens inside
- Usable even with new insides

Examples of Data Organization







- Using an ADT allows us to generalize the idea of grouping objects as collections

Scenario

- Revisit your `Anytown` project and replace the existing contents in the `main()` method of the `AnytownTest` class so that it has the following functionality.
  - Define an array to store 11 `Building` objects
  - Add the data file ***buildings.txt*** to the `src` folder of your project)
  - Write code to read the descriptions of 11 buildings from the data file, one per line of data, and add them to the array. Each field in each line of data is separated by | characters. The initial field is the building type, where **H** denotes a `House` object, **S** denotes a `Shop` object and **B** is a general `Building` that is neither a house or shop. All buildings have address and owner fields, while `House` objects also have values for number of bedrooms and a true/false field indicating the presence of a garage, and `Shop` objects also have values for the number of employees and the average annual turnover (in £K)
  - When all buildings are created, print their descriptions to the console.