Interfaces

Abstract Definition

abstract: existing in thought or as an idea but not having a physical or concrete existence

abstract method: A method name, it's arguments, and return type, but no Java instructions. Here's a method that should exist, but doesn't have a concrete set of instructions yet.

Interface: Simple Definition

An interface is a list of abstract method definitions.

Defining Interfaces

- An interface, like a class, goes in it's own .java file
- Interface names (like a classes) start with uppercase letter
- Interface Syntax:

```
package pkg_name;

interface interface_name {
        abstract_method_declarations;
}
```

- abstract keyword is implied – not required in method declarations
- Warning: methods in interfaces must be public!

Implementing Interfaces

- Classes may implement one or more interfaces
- Implements syntax:

```
class class_name implements interface_name
{
        …
}
```

- If a class implements an interface, it is like inheriting all the abstract method definitions in that interface
- Either the class must be an abstract class

OR

- The class must contain concrete definitions for all the methods

Example Interface

Countable.java

```
package xmp_interface;

interface Countable {
    public boolean isEmpty();
    public int size();
}
```

Blocks.java

```
package xmp_interface;

class Blocks implements Countable {
    private ArrayList<Block> blocks;
    …
    public boolean isEmpty() {
        return blocks.isEmpty(); }
    public int size() {
        return blocks.size(); }
    …
}
```

Why Interfaces?

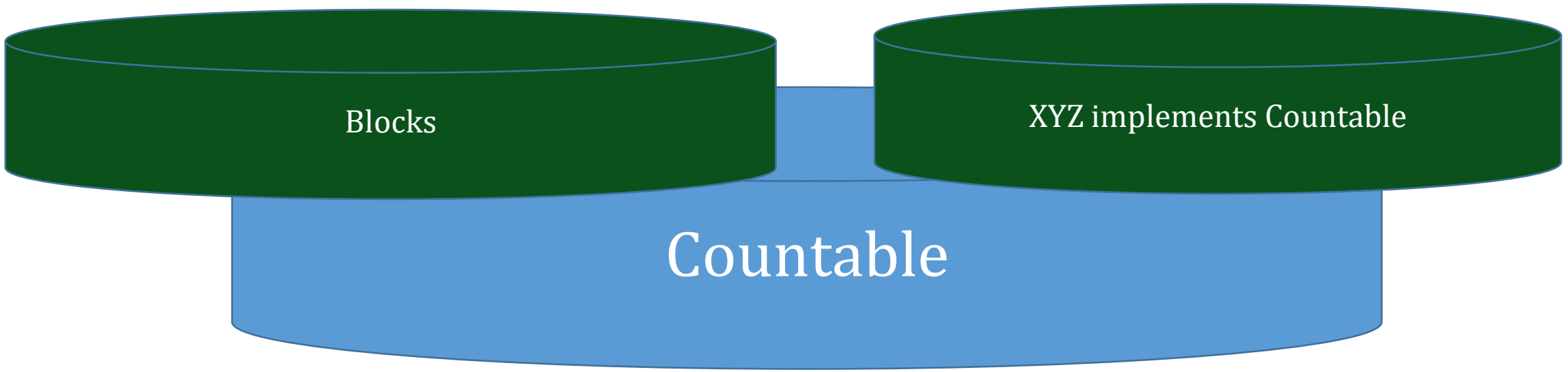
- Interfaces are more powerful than classes!
- Interfaces define capabilities (like an adjective)
  - If I know that a class implements an interface then I know at least some of what I can do to/on objects in that class
  - If I only use methods defined in the interface, I can create code which will work on ANY class which implements that interface!
- Interface standardizes common methods
  - If I learn "isEmpty()" for ArrayLists, I know "isEmpty()" for Blocks, …

Interfaces as Types

- You can declare a reference variable as a reference to an interface!
- You may assign that reference variable to any object in a class which implements that interface!
- You may invoke any methods defined in the interface.

- Interface acts like a sub-type of all classes which implement that interface

Interface as Type graphically



Interfaces as Abstractions

- An interface is an abstract view of a feature of a set of classes
  - Like an adjective which can modify multiple nouns
- Within these classes, there must be a concrete implementation of that feature
- Outside of these classes, we can write software which manages the abstraction
  - Only depends on the methods defined in the interface
  - Can be used on ANY object that implements that interface
- Gives us all the power of Object Orientation
  - inheritance, sub-typing, polymorphism, dynamic typing
- But using a simpler, more intuitive mechanism!