Abstract Classes
The Real Story?

- Standard
  - extends
  - NaturalNumber-Kernel
    - extends
    - Comparable
      - extends
    - NaturalNumber
      - implements
        - NaturalNumber1L
        - NaturalNumber2
The Real Story?

Does each of `NaturalNumber1L` and `NaturalNumber2` contain code for all 21 (yes, there are 21) methods that are introduced in, plus those inherited by, interface `NaturalNumber`? No!
The Ubiquitous Class: \texttt{Object}

- \textit{Every} class in Java extends \texttt{Object}, which is a special built-in class that provides default implementations for the following instance methods (among a few others that are not so important):
  
  \begin{verbatim}
  boolean equals(Object obj)
  int hashCode()
  String toString()
  \end{verbatim}
equals

boolean equals (Object obj)

• Reports whether this is equal to obj.
• Ensures:
  \[ equals = (this = obj) \]
equals

boolean equals(Object obj)
• Reports whether this is equal to obj.
• Ensures:
  
equals = (this = obj)

The default implementation in Object checks reference equality, though we expect object value equality! So, we (almost always) need to override this method.
hashCode

\texttt{int} \hspace{1em} \texttt{hashCode()} \\
\begin{itemize}
  \item Returns an \texttt{int} value that is functionally determined by the value of \texttt{this}.
  \item Ensures:
    \begin{align*}
    \text{hashCode} &= \text{[result of some fixed integer-valued function applied to this]}
    \end{align*}
\end{itemize}
hashCode

int hashCode()

- Returns an int value that is functionally determined by the value of this.
- Ensures:
  
  \[
  \text{hashCode} = \text{\{result of some fixed integer-valued function applied to this\}}
  \]

The default implementation in Object returns an int that depends on the reference value of this, though we expect it to depend on the object value! So, we (almost always) need to override this method.
**toString**

String toString()

• Returns the string representation of this.

• Ensures:

  ```
  toString = [the string representation of this]
  ```
String toString()

- Returns the string representation of this.
- Ensures:

  `toString = [the string representation of this]`

The default implementation in `Object` returns a `String` that shows the reference value of `this`, though we expect it to show the object value! So, we (almost always) need to override this method.
Abstract Classes

• Java permits you to write a kind of “partial” or “incomplete” class that contains bodies for some *but (typically) not all* of the methods of the interfaces it claims to implement.

• Such a class is called an *abstract class*:

```java
abstract class AC implements I {
    ...
}
```
Abstract Classes

• Java permits you to write a kind of "partial" or "incomplete" class that contains bodies for some but (typically) not all of the methods of the interfaces it claims to implement.

• Such a class is called an **abstract class**:

```java
abstract class AC implements I {
    ...
}
```

Because some methods still might not have bodies, Java will not let you *instantiate* an abstract class; that is, you cannot use an abstract class like a normal class and create a new object from it.
Example

```java
public abstract class NaturalNumberSecondary
    implements NaturalNumber {
    ...
}
```

---

15 May 2014

OSU CSE
public abstract class NaturalNumberSecondary implements NaturalNumber ...

This relationship is implicit: every class that does not extend some other class directly extends `Object`.

1. `NaturalNumber` implements `NaturalNumberSecondary`
2. `Object` extends `NaturalNumberSecondary`
3. `Object` extends `NaturalNumber`
4. `NaturalNumberSecondary` implements `NaturalNumber`
This abstract class has code that overrides the default implementations (inherited from Object) of equals, hashCode, and toString for NaturalNumbers, so they do “the right thing”, i.e., so their behaviors are based on object values rather than reference values

– Details of this code later... see “Resources”
NaturalNumberSecondary

• This abstract class also has code that implements all the methods *introduced* in the NaturalNumber interface, but not those inherited by it from other interfaces
  – Details of this code later... see “Resources”
  – Note that this still leaves the methods introduced in the Standard and NaturalNumberKernel interfaces without bodies; hence, it’s an abstract class
The Bigger Picture: Old Story

- **NaturalNumber**
  - **NaturalNumber1L**
  - **NaturalNumber2**
- **NaturalNumberKernel**
  - **Standard**
  - **Comparable**
The Bigger Picture: New Story

The new abstract class plus `Object`, in context...
The Bigger Picture: New Story

- newInstance
- clear
- transferFrom
- constructors
  - multiplyBy10
  - divideBy10
  - isZero
- increment
  - ...
- root

- NaturalNumber extends
  - NaturalNumberKernel
  - Standard
  - Comparable
  - Object

- NaturalNumberSecondary
  - extends
  - NaturalNumber1L
  - NaturalNumber2
The Kernel Classes

- This leaves the kernel classes (in the example, \texttt{NaturalNumber1L} and \texttt{NaturalNumber2}) with only a few things left to implement, i.e., the 4 constructors plus the 6 methods introduced in:
  - Standard (i.e., \texttt{newInstance}, \texttt{clear}, and \texttt{transferFromFile})
  - \texttt{NaturalNumberKernel} (i.e., \texttt{multiplyBy10}, \texttt{divideBy10}, and \texttt{isZero})
Layered Methods

Has *layered* method bodies for `equals`, `hashCode`, and `toString` (overriding those methods from `Object`) ...
Layered Methods

... and *layered* method bodies for all 12 methods introduced in \textbf{NaturalNumber}.
Kernel Methods

Has bodies for 4 constructors plus 6 methods introduced in Standard and NaturalNumberKernel.
Kernel Methods

Has bodies for 4 constructors plus 6 methods introduced in Standard and NaturalNumberKernel.
Factoring Out Common Code

• Method bodies that can be written once—and work for any implementation of `NaturalNumberKernel` because they are *programmed to that interface*—have been factored out into an abstract class

• This leaves only constructors and a few kernel methods to be implemented in `NaturalNumber1L`, `NaturalNumber2`, and future kernel classes (if any)
Factoring Out Common Code

• Method bodies that can be written once—and work for any implementation of `NaturalNumberKernel` because they are *programmed to that interface*—have been *factored out into an abstract class*

• This leaves only constructors and a few kernel methods to be implemented in `NaturalNumber1L`, `NaturalNumber2`, and future kernel classes (if any)

This is a **best practice** use for abstract classes.
Another Kettle of Fish

• The code in each *kernel class* (e.g., in the example `NaturalNumber1L` and `NaturalNumber2`) implements only 4 constructors and 6 methods each, not all 21 methods of `NaturalNumber`.

• But, implementing these few methods is different than implementing the other 15 *layered* methods where you can *call* the kernel methods to do the work!
Resources

• OSU CSE Components API: NaturalNumber
  – http://cse.osu.edu/software/common/doc/