Quick Review of OOP Constructs

- **Classes:**
  - Data types for structured data and behavior
  - fields and methods

- **Objects:**
  - Variables whose data type is a class

- **Fields:**
  - Variables of an object that store data

- **Methods:**
  - Parts of an object that define instructions

- **Constructors:**
  - Special methods that initialize instance fields
Constructors

- These are special methods that are most commonly used to initialize instance fields.
- Definitions of constructors must obey these rules:
  - The name of the constructor must match (exactly) the name of the class.
  - The constructor must not be given any return type, and it must not contain any return statement.
  - If the class extends another class that has no zero-argument constructor, then the first line of the constructor must be a call to the `super(<args>)` constructor.

- Constructors are called differently from other methods, by using the `new` keyword:
  ```java
  MyClass myObject = new MyClass(<some args>);
  ```
Instance vs. static fields

- If a field is not declared to be static, it is an instance field.
- To decide if a field should be static or instance, consider:
  - Does each object potentially have a different value for that field?
    - If so, the field should be instance.
    - For example, BankAccount objects might all have different values for a balance field.
  - Do all objects share the same value for that field?
    - If so, the field should be static.
    - For example, all BankAccount objects might share the same value for the interest rate. So the interestRate field should be declared static.
- If a class defines an instance field, every object (or variable) has a separate location in memory to store data for that field.
- If a class defines a static field, every object has a reference to a single location in memory, shared by all objects of that class, to store data for that field.
Instance vs. static methods

- Instance and static methods differ in how they are called:
  - To call an instance method: `myObject.instanceMethod();`
  - To call a static method: `MyClass.staticMethod();`
  - Technically, it is OK to call a static method using an object rather than a class name. The compiler will give you a warning, but not an error.
  - The opposite is an error: to call an instance method, you **MUST** supply an object name.

- Instance methods have an *implicit parameter*, the object which was used to call the method. Static methods have no implicit parameter.
  - In the example above, `myObject` becomes the implicit parameter for `instanceMethod()`.
  - Instance methods can use the `this` keyword to refer to the implicit parameter.

- Static methods CANNOT refer to instance fields, or to the `this` keyword.
  - Because static methods don’t have an implicit parameter, `this` doesn’t mean anything inside the body of a static method.
  - However, it is OK for instance methods to refer to static fields.
OOP Review

- **Access specifiers:**
  - **public:** any class can refer to it
  - **private:** only the class that defines it can refer to it
  - **protected:** the class that defines it, and any direct or indirect subclass, can refer to it
  - **<no specifier>:** package-level access, meaning that any class in the same directory can refer to it

- **Specifiers can be used for:**
  - Classes or interfaces
  - Methods
  - fields
Reference Semantics

- Object variables in Java are references to locations in memory
- Shallow copies (via assignment or parameter passing) create duplicate references, not duplicate objects
- If the data of one object reference is changed, it changes the data of any other reference to the same location
- Primitive types (int, short, double, etc.) do not use reference semantics
Inheritance

- Subclasses can extend superclasses
  - A subclass inherits all the fields and methods of the superclass
- Subclasses can override definitions in the superclass
  - If a subclass defines a new version of an inherited method or field, it overrides the superclass’s version

Final

- Final methods and fields can’t be overridden
- Final classes can’t be extended

Abstract

- Abstract methods must be overridden to be usable
- If a class contains an abstract method, it must be abstract
- If a class is abstract, it can’t have a constructor
OOP Review

- Interfaces
  - Much like an abstract class that has all abstract methods
  - It declares, but does not define, a set of methods
  - Classes that define the methods *implement* the interface
Polymorphism

- A variable of a class type can hold an object of that type or any indirect subclass type.
- A variable of an interface type can hold an object of any implementing type.
- The behavior of a method call on a class or interface variable depends on the run-time type of the object that the variable is a reference to.
**Identifiers: Keywords**

- **keyword**: An identifier that you cannot use, because it already has a reserved meaning in the Java language.

- Complete list of Java keywords:

<table>
<thead>
<tr>
<th>Abstract</th>
<th>Default</th>
<th>If</th>
<th>Implements</th>
<th>Private</th>
<th>This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Do</td>
<td>Import</td>
<td>Interface</td>
<td>Protected</td>
<td>Throws</td>
</tr>
<tr>
<td>Break</td>
<td>Double</td>
<td>Instanceof</td>
<td>Int</td>
<td>Public</td>
<td>Transient</td>
</tr>
<tr>
<td>Byte</td>
<td>Else</td>
<td>Instanceof</td>
<td>Long</td>
<td>Return</td>
<td>Try</td>
</tr>
<tr>
<td>Case</td>
<td>Extends</td>
<td>Interface</td>
<td>Native</td>
<td>Short</td>
<td>Try</td>
</tr>
<tr>
<td>Catch</td>
<td>Final</td>
<td>Interface</td>
<td>New</td>
<td>Static</td>
<td>Void</td>
</tr>
<tr>
<td>Char</td>
<td>Finally</td>
<td>Long</td>
<td>New</td>
<td>Strictfp</td>
<td>Volatile</td>
</tr>
<tr>
<td>Class</td>
<td>Float</td>
<td>Native</td>
<td>New</td>
<td>Super</td>
<td>While</td>
</tr>
<tr>
<td>Const</td>
<td>For</td>
<td>Package</td>
<td>Synchronized</td>
<td>Switch</td>
<td></td>
</tr>
<tr>
<td>Continue</td>
<td>Goto</td>
<td>Package</td>
<td>Synchronized</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>