News

- Homework 3 is online
  - Due this Friday (December 4)

- Final exam: Dec 11, 8:30 am, OSBO A
  - Length: 2:30
Reading Assignments

- Reading for this week: inheritance
  - Edition 3: Ch 10
  - Edition 2: Ch 13

- Optional reading: graphics, UI
  - Edition 3: Ch 2.11, 2.12, 9.5-9.8, 10.9, 10.10
  - Edition 2: Ch 5.1, 5.2, 11.5, 12.1-12.3
  - (NOT part of the final exam…)
Recap: Abstract Classes

- Serve as placeholders in class hierarchy
  - Contain a partial description that is inherited by all its descendants
  - These descriptions are not useful in itself
    - Some methods are not implemented
    - Thus, the class cannot be instantiated

- A class becomes abstract by including the `abstract` modifier in the class header
Recap: Abstract Classes – Another View

- Abstract classes are not completely defined
  - Contain one or more abstract methods
    - Just the header (signature) is given, but not implementation
    - Children need to implement the method
  - Abstract methods are used when it is not fully possible to provide an implementation that makes sense across class hierarchy
    - E.g. `timeStep` method in our example
Recap: Abstract Classes vs. Interfaces

- Could we use an interface instead to describe TimeVaryingObject?
  - Yes, in this the specific example
  - But remember: interfaces cannot provide ANY implementations
  - Abstract classes can have SOME methods that are implemented
    - And these get inherited by the children....
Recap: So Why do we Need Interfaces?

- Can’t we do everything with abstract classes instead?
- Well not really:
  - Java does not allow for multiple inheritance, but it allows for a class to implement multiple interfaces
Recap: Multiple Inheritance vs. Multiple Interfaces

- Multiple inheritance can be useful in languages like C++, but it makes things messy
  - E.g. what if both parent classes have a method with the same signature? Which is visible in child?
- Why is this not an issue with interfaces?
  - Interfaces don’t actually provide implementations, so there is no conflict
  - If two interfaces specify the same method, they share one implementation
Recap: Interfaces vs. Abstract Classes

- An interface differs from an abstract class in the following ways:
  - An abstract class is an incomplete class that requires further specialization. An interface is just a specification or prescription for behavior.
  - An interface doesn't have any overtones of specialization that are present with inheritance. It merely says, "We need a certain method that users should be able to call."
  - A class can implement several interfaces at once, whereas a class can extend only one parent class.
Recap: Interfaces vs. Abstract Classes

- Advanced notes:
  - An interface can extend another interface. It can’t extend an abstract class or a "concrete" class.
  - While a class can legally implement an interface, it might only implement some of the methods prescribed by the interface. If so, the class becomes an abstract class that must be further extended (inherited) before it can be instantiated.
Recap: Who Can do What?

- An interface can be implemented only by a class or an abstract class.
- An interface can be extended only by another interface.
- A class can be extended only by a class or an abstract class.
- An abstract class can be extended only by a class or an abstract class.
- Interfaces are not classes and can't be instantiated. Abstract classes may have undefined methods and can't be instantiated.
Objectives

- Practice inheritance of existing classes
- Learn about basic graphics constructs in Java
Java Graphics

- Java systems provide lots of classes for user interfaces and graphics
  - These are not strictly part of the language, but they are available in almost all Java implementations
  - We will learn how to use some of these classes, and in the process practice inheritance
Creating a Window in Java

- Java JFrame class:
  - Part of the Swing package for user interfaces
  - Can create a window, name it, size it, etc.
Drawing Stuff

- How do we actually show something in our window?
  - Have to add objects of type JComponent to our frame
  - JComponent is an abstract base class for all kinds of drawable objects
  - Have to create subclasses that extend JComponent, and implement method void paintComponent(Graphics g)
The parameter to the `paintComponent` method is an object of type `Graphics`.

- Actually, in our case it will be providing a reference to a `Graphics2D` object.
- `Graphics` is an abstract base class.
- `Graphics2D` is a subclass of `Graphics`.
- The classes describe a `graphics context`, i.e. a collection of state that is required to draw stuff (i.e. window area, color, etc...)
Drawing Stuff

- Using the Graphics2D object, we can now draw geometric primitives
  - E.g: Line2D.Double
  - Line2D.Double is a nested class (class within a class)…
Let’s Use this to draw a Snow Flake

- Create Snowflake class (subclass of JComponent)
  - Snowflake has a position and a radius…
- Right now, our snowflake is just a hexagon
  - We’ll make this prettier next time…