Announcements

- Regarding TA Room Usage
  - No food or drinks are allowed in the TA room
  - Please do not rearrange the furniture
  - No independent studying (not a study lounge)
  - Please be considerate of fellow students who need help. Once you have spoken with your TA please clear out to allow other students the same opportunity

- Link with information at
Comparator Interface

- Comparator
  - public int compare(T a, T b)
    - Negative if a < b, 0 if a == b, positive if a > b
- Properties
  - Imposes total ordering on objects of a class
  - Provide alternatives to natural ordering
  - Supports generics
    - Example: `class myC implements Comparator<Foo>{ … }`
  - Use as parameter for sort function
    - Example: `Collections.sort(myFooList, new myC( ) );`
- Example: `comparatorExample`
Three Levels of Copying Objects

Assume y refers to object z

1. Reference copy
   - Makes copy of reference
   - $x = y$;

2. Shallow copy
   - Makes copy of object
   - $x = y.clone()$;

3. Deep copy
   - Makes copy of object z and all objects (directly or indirectly) referred to by z
Cloning

- Cloning
  - Creates identical copy of object using clone( )
- Cloneable interface
  - Supports clone( ) method
  - Returns copy of object
    - Copies all of its fields
    - Does not clone its fields
    - Makes a shallow copy
- Example: cloning package
Garbage Collection

- Concepts
  - All interactions with objects occur through reference variables
  - If no reference to object exists, object becomes garbage (useless, no longer affects program)

- Garbage collection
  - Reclaiming memory used by unreferenced objects
  - Periodically performed by Java
  - Not guaranteed to occur
  - Only needed if running low on memory
Destructor

• Description
  • Method with name `finalize()`
  • Returns void
  • Contains action performed when object is freed
  • Invoked automatically by garbage collector
    • Not invoked if garbage collection does not occur
  • Usually needed only for non-Java methods

• Example
  ```java
  class Foo {
    void finalize() { … } // destructor for foo
  }
  ```
Initialization Block

• Definition
  • Block of code used to initialize static & instance variables for class

• Motivation
  • Enable complex initializations for static variables
    • Control flow
    • Exceptions
  • Share code between multiple constructors for same class
Initialization Block Types

- Static initialization block
  - Code executed when class loaded
- Initialization block
  - Code executed when each object created
  - (at beginning of call to constructor)
- Example

```java
class Foo {
    static {
        A = 1;
    } // static initialization block
    {
        A = 2;
    } // initialization block
}
```
Variable Initialization

- Variables may be initialized
  - At time of declaration
  - In initialization block
  - In constructor

- Order of initialization
  1. Declaration, initialization block
     (in the same order as in the class definition)
  2. Constructor
Variable Initialization – Example

class Foo {
    static { A = 1; }  // static initialization block
    static int A = 2;  // static variable declaration
    static { A = 3; }  // static initialization block
    { B = 4; }         // initialization block
    private int B = 5; // instance variable declaration
    { B = 6; }         // initialization block
    Foo() {            // constructor
        A = 7;
        B = 8;
    }                  // now A = 7, B = 8
}                       // initializations executed in order of number
public class Person {

    // STATIC INITIALIZATION CREATES OBJECT ONCE
    private static final Date MILLENIUM;
    static {
        Calendar gmtCal = Calendar.getInstance(
            TimeZone.getTimeZone("GMT"));
        gmtCal.set(2000,Calendar.JANUARY,1,0,0,0);
        Date MILLENIUM = gmtCal.getTime();
    }

    public boolean bornBefore2000(){ // FASTER!
        return birthDate.before(MILLENIUM);
    }
}