

CMSC 132: OBJECT-ORIENTED PROGRAMMING II



Java Language Constructs II

Department of Computer Science
University of Maryland, College Park

Regarding Questions over E-mail

- Due to the large number of students in class, we (instructors and TAs) cannot address project questions, questions about lecture material, etc. over e-mail. If you have any questions, please address them in lab/discussion session, in lecture or during office hours. Thank you for your cooperation.

About Quizzes/Exams

- Please read the guidelines at:

<http://www.cs.umd.edu/~nelson/classes/utilities/examRules.html>

About Style

- Let's go over the following information

<http://www.cs.umd.edu/class/spring2012/cmsc132/resources/StyleGuidelines.html>

Implementing Equals

- Approach we want to use (assuming class **A**)

```
public boolean equals(Object obj) {  
    if (obj == this)  
        return true;  
    if (!(obj instanceof A)) // covers obj == null case  
        return false;  
    A a = (A)obj;  
    /* Specific comparison based on A fields appears here */  
}
```

- What happens if we use comparisons of Class objects rather than instanceof?
- Example: equalsMethod package

Comparable Interface

- Comparable
 - `public int compareTo(T o)`
 - `a.compareTo(b)` returns
 - **Negative** if $a < b$, **0** if $a == b$, **positive** if $a > b$
- Properties
 - Referred to as the class's *natural ordering*
 - Can sort using `Collections.sort()` & `Arrays.sort()`
 - Example: **`Collections.sort(myList);`**
 - Can use as keys in `SortedMap` & `SortedSet`
 - Consistency w/ `equals()` strongly recommended
 - `x.equals(y)` if and only if `x.compareTo(y) == 0`
- Example: `comparableExample` package

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`

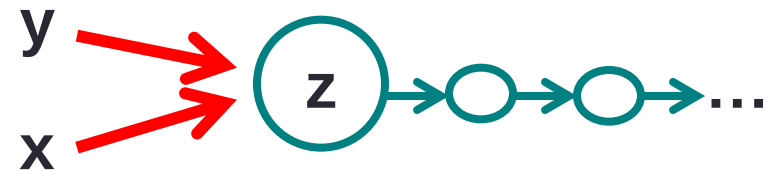
Three Levels of Copying Objects

Assume y refers to object z

1. Reference copy

- Makes copy of reference

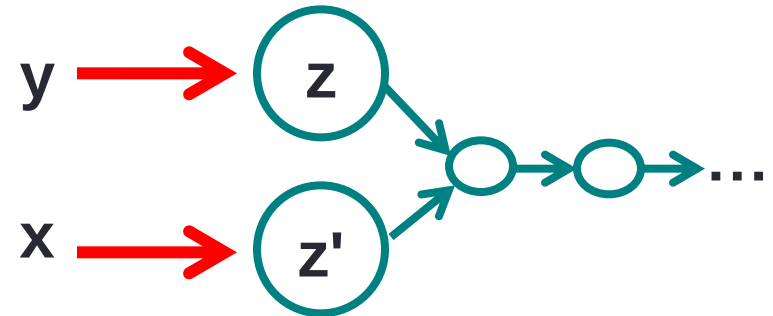
- `x = y;`



2. Shallow copy

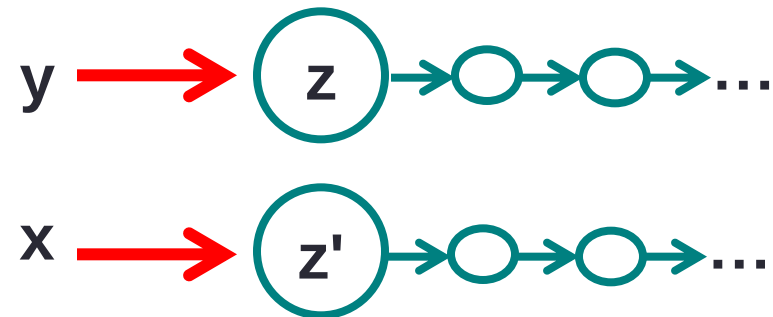
- Makes copy of object

- `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

```
class Foo {  
    void finalize() { ... }           // destructor for foo  
}
```

Annotations

- Annotation → Java construct that allow us to add validity constraints to Java Classes
- Validity constraint example
 - A instance variable cannot assume a negative value
 - A parameter can not be null
 - A method in a class must override a method in its superclass
- Syntax

at-sign (@) followed by annotation type and a parenthesized list of element-value pairs
- Example

`@DefaultAnnotationForParameters(NonNull.class)`
- You can ignore annotations in code distribution for class projects