

## Objectives

- Packaging
- Collections
- Generics
- Eclipse

## Iteration over Code

- Assignment 5
- Demonstrates typical design/implementation process
  - Start with your original code design
    - Inheritance from GamePiece class
  - Realize it could be designed better
    - Make GamePiece class abstract
    - Use an array of GamePiece objects
    - Easier to add new functionality to Game
- Major part of problem-solving is figuring out how to break problem into smaller pieces

## Review

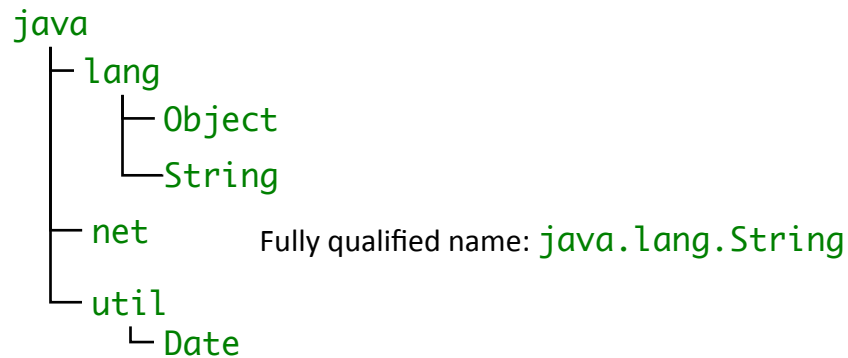
- How do we specify that a class or a method cannot be subclassed/overridden?
- Compare and contrast abstract classes and interfaces
- When should a class be abstract?
  - If you extend an abstract class, do you have to override all abstract methods?
- When should you create/use an interface?
- What is the keyword for specifying that your class adheres to an interface?

## PACKAGES

## Packages

- Hierarchical structure of Java classes

- Directories of directories



- Use **import** to access packages

Sept 28, 2016

Sprenkle - CSCI209

5

## Standard Practice

- To reduce chance of a conflict between names of classes, put classes in **packages**
- Use **package** keyword to say that a class belongs to a package:
  - `package java.util;`
  - *First line in class file*
- Typically, use a unique prefix, similar to domain names
  - `com.ibm`
  - `edu.wlu.cs.logic`

Sept 28, 2016

Sprenkle - CSCI209

6

## Importing Packages

- Can import one class at a time or all the classes within a package

- Examples:

```
import java.util.Date;  
import java.io.*; ← Import entire package
```

- \* form may increase compile time
  - BUT, no effect on run-time performance

## COLLECTIONS

## Collections

- Sometimes called *containers*
- Group multiple elements into a single unit
- Store, retrieve, manipulate, and communicate aggregate data
- Represent data items that form a natural group
  - Poker hand (a collection of cards)
  - Mail folder (a collection of messages)
  - Telephone directory (a mapping of names to phone numbers)

Sept 28, 2016

Sprenkle - CSCI209

9

## Java Collections Framework

- *Unified architecture* for representing and manipulating collections
- More than arrays
  - More flexible, functionality, dynamic sizing
- `java.util`

Sept 28, 2016

Sprenkle - CSCI209

10

## Collections Framework

- **Interfaces**

- Abstract data types that represent collections
- Collections can be manipulated *independently* of implementation

- **Implementations**

- Concrete implementations of collection interfaces
- Reusable data structures

- **Algorithms**

- Methods that perform useful computations on collections, e.g., searching and sorting
- Reusable functionality
- **Polymorphic**: same method can be used on many different implementations of collection interface

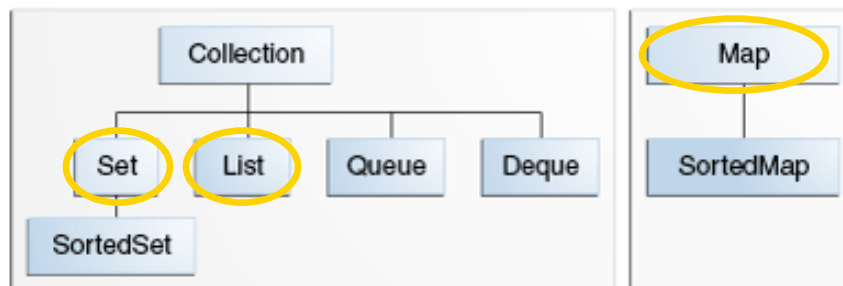
Sept 28, 2016

Sprenkle - CSCI209

11

## Core Collection Interfaces

- Encapsulate different types of collections



Sept 28, 2016

Sprenkle - CSCI209

12

# GENERICS

Sept 28, 2016

Sprenkle - CSCI209

13

## Example of the Way It Was

- Before Java 1.5
- Doesn't know what *type* of data is in the List

```
List myList = new LinkedList();
myList.add(new Card(4, "clubs"));
...
Card x = (Card) myList.get(0);
```

Returns an Object ←

- Have to cast object we get out of list to desired type
- What if someone put in an object of wrong type previously?
- Have similar issue in Python

Sept 28, 2016

Sprenkle - CSCI209

14

## Generic Collection Interfaces

- Added to 1.5
- Declaration of the Collection interface:

```
public interface Collection<E>...
```

Type parameter

- <E> means interface is generic for element class
- When declare a Collection, **specify type** of object it contains
  - Make sure put in, get out appropriate type
  - Allows compiler to verify that object's type is correct
    - Reduces errors at runtime
- Example, a hand of cards: Always declare type

```
List<Card> hand = new ArrayList<Card>();
```

**New in Java 7:** List<Card> hand = new ArrayList<>();

Sept 28, 2016

## Comparable Interface

- Also uses Generics

```
public interface Comparable<T>
```

The type it compares

```
int compareTo(T o)
```

Sept 28, 2016

Sprenkle - CSCI209

Chicken.java

16



## Comparing: Before & After Generics

- Before Generics

```
List myList = new LinkedList();  
myList.add(new Card(4, "clubs"));  
...  
Card x = (Card) myList.get(0);
```

- After Generics

```
List<Card> myList = new LinkedList<>();  
myList.add(new Card(4, "clubs"));  
...  
Card x = myList.get(0);
```

✓ Improved readability and robustness

## Types Allowed with Generics

- Can only contain **Objects**, not primitive types
  - **Autoboxing and Autounboxing to the rescue!**
    - Example: If collecting **ints**, use **Integer**

## WRAPPER CLASSES

Sept 28, 2016

Sprenkle - CSCI209

19

## Wrapper Classes

- **Wrapper class** for each primitive type
- Sometimes need an instance of an Object
  - To store in Lists and other Collections
- Include functionality of parsing their respective data types

```
int x = 10;  
Integer y = new Integer(10);
```

Sept 28, 2016

Sprenkle - CSCI209

20

## Wrapper Classes

- **Autoboxing** – automatically create a wrapper object

```
// implicitly 11 converted to
// new Integer(11);
Integer y = 11;
```

- **Autounboxing** – automatically extract a primitive type

```
Integer x = new Integer(11);
int y = x.intValue();
int z = x; // implicitly, x is x.intValue();
```

Convert right side to whatever is needed on the left

Sept 28, 2016

Sprenkle - CSCI209

21

## Effective Java: Unnecessary Autoboxing

- Can you find the inefficiency from object creation?

```
Long sum = 0L;
for (long i=0; i < Integer.MAX_VALUE; i++) {
    sum += i;
}
System.out.println(sum);
```

- How to fix?

Sept 28, 2016

Sprenkle - CSCI209

[Autobox.java](#)

22

## Effective Java: Unnecessary Autoboxing

- Can you find the inefficiency from object creation?

```
Long sum = 0L;
for (long i=0; i < Integer.MAX_VALUE; i++ ) {
    sum += i;           Constructs 231 Long instances
}
System.out.println(sum);
```

- How to fix?

Sept 28, 2016

Sprenkle - CSCI209

Autobox.java

23

## Effective Java: Unnecessary Autoboxing

- Can you find the inefficiency from object creation?

```
Long sum = 0L;
for (long i=0; i < Integer.MAX_VALUE; i++ ) {
    sum += i;           Constructs 231 Long instances
}
System.out.println(sum);
```

- How to fix?
- Lessons:
  - Prefer primitives to boxed primitives
  - Watch for unintentional autoboxing

Sept 28, 2016

Sprenkle - CSCI209

Autobox.java

24

# LISTS

Sept 28, 2016

Sprenkle - CSCI209

25

## List

- An *ordered* collection of elements
- Can contain duplicate elements
- Has control over where objects are stored in the list

Sept 28, 2016

Sprenkle - CSCI209

26

## List Interface

- **boolean** `add(<E> o)`
  - Boolean so that List can refuse some elements
    - e.g., refuse adding **null** elements
- **<E>** `get(int index)`
  - Returns element at the position index
  - Different from Python: no shorthand
    - Can't write ~~`list[pos]`~~
- **int** `size()`
  - Returns the number of elements in the list
- And more!
  - `contains`, `remove`, `toArray`, ...

Sept 28, 2016

Sprenkle - CSCI209

27

## Common List Implementations

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>● <b>ArrayList</b> <ul style="list-style-type: none"> <li>➤ Resizable array</li> <li>➤ Used most frequently</li> <li>➤ Fast</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>● <b>LinkedList</b> <ul style="list-style-type: none"> <li>➤ Use if adding elements to ends of list</li> <li>➤ Use if often delete from middle of list</li> <li>➤ Implements <code>Deque</code> and other methods so that it can be used as a stack or queue</li> </ul> </li> </ul> |
|---|--|

How would you find the other implementations of List?

Sept 28, 2016

Sprenkle - CSCI209

28

## Implementation vs. Interface

Implementation choice only affects performance

- Preferred Style:

1. Choose an implementation
2. Assign collection to variable of corresponding **interface** type

```
Interface variable = new Implementation();
```

- Methods should accept interfaces—not implementations

Why is this the preferred style?

Sept 28, 2016

Sprenkle - CSCI209

29

## Implementation vs. Interface

Implementation choice only affects performance

- Preferred Style:

1. Choose an implementation
2. Assign collection to variable of corresponding **interface** type

- Why?

- Program does not depend on a given implementation's methods
  - Access only using interface's methods
- Programmer can change implementations
  - Performance concerns or behavioral details

Sept 28, 2016

Sprenkle - CSCI209

30

## Discussion of Deck Class

`cards.Deck.java`

Sept 28, 2016

Sprenkle - CSCI209

31

## SETS

Sept 28, 2016

Sprenkle - CSCI209

32



## Set Interface

- No duplicate elements
  - Needs to determine if two elements are “logically” the same (`equals` method)
- Models mathematical set abstraction

Sept 28, 2016

Sprenkle - CSCI209

33

## Set Interface


- `boolean add(<E> o)`
  - Add to set, only if not already present
- `int size()`
  - Returns the number of elements in the list
- And more! (`contains`, `remove`, `toArray`, ...)
  - Note: no `get` method -- get #3 from the set?

Sept 28, 2016

Sprenkle - CSCI209

34

## Some Set Implementations

- **HashSet** 
  - Implements set using *hash table*
    - *add*, *remove*, and *contains* each execute in  $O(1)$  time
  - Used more frequently
  - Faster than **TreeSet**
  - No ordering
- **TreeSet**
  - Implements set using a *tree*
    - *add*, *remove*, and *contains* each execute in  $O(\log n)$  time
  - **Sorts**

## FindDuplicates Problem

- From the array of command-line arguments, identify the duplicates

```
public static void main(String args[]) {
}
}
```

## FindDuplicates

```
public static void main(String args[]) {
    Set<String> s = new HashSet<String>();
    for (String a : args) {
        if (!s.add(a)) {
            System.out.println(
                "Duplicate detected: " + a);
        }
    }
    System.out.println(s.size() +
        " distinct words detected: " + s);
}
```

How much does code changes if S is a TreeSet?



## 2011 Software System Award

*The Software System Award is given to an institution or individuals recognized for developing software systems that have had a lasting influence, reflected in contributions to concepts and/or commercial acceptance.*

created by IBM.

Eclipse changed the way builders think about tools by defining a set of user interaction paradigms for which domain-specific variants are plugged in and customized for their tool.

Conceived to address perceived shortcomings in proprietary software development tools, Eclipse allowed developers to seamlessly integrate their own extensions, specializations, and personalizations. ...

Sept 28, 2016

Sprenkle - CSCI209

39

## 2011 Software System Award

It revolutionized the notion of an Integrated Development Environment (IDE) by identifying the conceptual kernel underlying any IDE.

Eclipse was designed as an open, extensible platform for application development tools with a Java IDE built on top. In 2004 Eclipse became a not-for-profit corporation.

The IBM Eclipse team included John Wiegand, Dave Thomson, Gregory Adams, Philippe Mulet, Julian Jones, John Duimovich, Kevin Haaland, Stephen Northover (now with Oracle), and **Erich Gamma** (now with Microsoft).

Sept 28, 2016

Sprenkle - CSCI209

40



<http://www.eclipse.org/>

- Open source integrated development environment (IDE) for Java
- Has market share for Java IDEs
- Described as “an open extensible IDE for anything and nothing in particular”
- Provides a robust Java development environment
- Incorporates popular software development tools like JUnit and CVS
  - [More on those later this semester](#)
- Plugins allow extensibility

Sept 28, 2016

Sprenkle - CSCI209

41

## Project/Code Organization

- **workspace** directory contains all projects
  - Located in your home directory, unless you specified otherwise
- Use **projects** to organize your code
- Within a project
  - **src/** directory contains **.java** files
  - **bin/** directory contains **.class** files
    - Often hidden in GUI

Sept 28, 2016

Sprenkle - CSCI209

42

## Java Made Easier

- Creating class's basic functionality
  - See **Source** and **Refactor** menus
- Gives you a list of methods for an object
  - After you type **object.**
  - Then shows parameters for methods
- Automatically creates template of Javadoc
  - When you type **/\*\***
- Autocompletion of variables, methods
- Formatting code ...
- Shows unused fields/variables
- Shows compiler errors
- ...

Sept 28, 2016

Sprenkle - CSCI209

43

## Eclipse Demo

Why can a Java IDE provide this functionality?

- Show **BirthDay** class
  - Override **equals** and **toString** methods
- Create a new class
  - Generate **main** method, Comments
    - Create a String object, see methods used
- Demonstrate refactoring
  - Rename a field
  - Extract a method (month name)
- Run the Birthday Class (main)
  - Command line arguments

Sept 28, 2016

Sprenkle - CSCI209

44

## Eclipse Hints

- After you have written a method, type

`/**`

before the method, and then hit enter and the Javadocs template will be automatically generated for you

- Use command-spacebar for possible completions

## Installing at Home

- Go to [www.eclipse.org](http://www.eclipse.org)
- Select “Downloads”
  - Get the Eclipse Installer for Eclipse Neon
- Then “Eclipse IDE for Java **EE** Developers”
  - For developing web applications and other *enterprise* applications
  - We’re using Java 8 for the code we write
  - May need to install more than one version of Java to run

## Eclipse Discussion

- Helpful hints
  - Control-spacebar
  - Format the file
  - Auto-templates for Javadocs

Sept 28, 2016

Sprenkle - CSCI209

47

## Looking Ahead

- Assignment 6 – due *next* Wednesday
  - Multiple components
    - Eclipse practice
    - Collections
    - Generics
    - Packages
    - ...
  - Keep building, based on what we're doing in class

Sept 28, 2016

Sprenkle - CSCI209

48