Objectives

- Streams
- Enumerated types
- Collections

Review

- What are some ways to categorize streams?
  - When would you use these streams?

Scanners

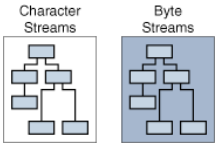
- Do not throw `IOExceptions`!
  - For a simple console program, `main()` does not have to deal with or throw `IOExceptions`
  - Required with `BufferedReader /InputStreamReader` combination
- Breaks its input into tokens using a delimiter pattern, which matches whitespace
  - What is "delimiter pattern"?
  - What is "whitespace"?
- Converts resulting tokens into values of different types using `nextXXX()`

Scanners

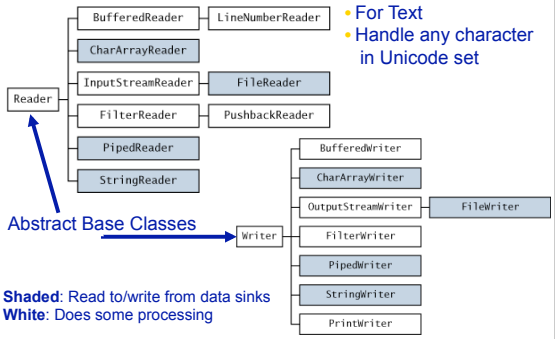
- Throws `InputMismatchException` when token does not match pattern for expected type
  - e.g., `nextLong()` called with next token "AAA"
  - `RuntimeException` (no catching required)
- Can change token delimiter from default of whitespace
- Assumes numbers are input as decimal
  - Can specify a different radix
- Scanners are for Java 1.5 and up only

`java.io` Classes Overview

- Two types of stream classes
  - Based on datatype: Character, Byte

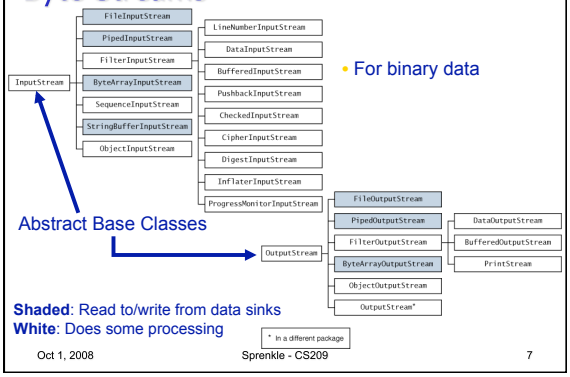


Character Streams



- For Text
- Handle any character in Unicode set

Byte Streams



- For binary data

Abstract Base Classes

Shaded: Read to/write from data sinks
White: Does some processing

Readers and Input Streams

Similar APIs for different data types

characters	bytes
Reader	InputStream
int read()	int read()
int read(char cbuf[])	int read(byte cbuf[])
int read(char cbuf[], int offset, int length)	int read(byte cbuf[], int offset, int length)

Writers, OutputStreams are similarly parallel

 PetSurvey.java

java.nio.*

- Additional classes for I/O
  - Scalable I/O
  - Fast buffered byte and character I/O
  - Character set conversion
- Designed for performance tuning

Additional I/O Functionality

- Java provides classes so that you can
  - Lock files (java.nio.channels.FileLock)
    - Coordinates accesses to files
      - Multiple programs read/write same file
    - Depends on OS to enforce locks
  - Read from random points in the file
    - java.io.RandomAccessFile

PARSING FILES

Parsing Files

- Use programs to automate tasks
- Often have large amounts of data in files
- Java provides classes to make parsing easier

Use String class

```
String[] split(String regex)
```

\\s means whitespace

```
String test = "this is a test";
String[] result = test.split("\\s");
for (int x=0; x<result.length; x++)
    System.out.println(result[x]);
```

Output:
```
this
is
a
test
```

We also used " " to split on a space

Regular Expressions in Java

Predefined character classes

Class	Meaning
.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

- Regular expressions are very powerful
- We'll discuss regular expressions more later

StreamTokenizer

- Tokenize an incoming character stream
- Table-driven lexical analyzer
  - Every possible input character has a significance
  - Scanner uses the significance of the current character to decide what to do
- Compiler terminology!
- May be useful to parse files
  - E.g., handling comments like /* */ or //

ENUMERATED TYPES

Enumerated Types

- Also called *enums*
  - More powerful than enums in C
  - New to Java 1.5
- Are like **inner** classes in Java
  - Entirely nested within another class
- Implicitly inherits from `java.lang.Enum`
- Example:

```
enum Season { Spring, Summer, Fall, Winter };
Season now = Season.Fall;
```


Enums

- Has static `values()` method
  - Returns array of values in order declared
  - E.g., Spring, Summer, Fall, Winter
- Can add functionality to enum
  - Add methods
- Can be used in `switch` statements

`cards.Card.java`

COLLECTIONS

Collections

- Also known as *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 letters)
  - Telephone directory (a mapping of names to phone numbers).
- Examples: `HashMaps, Sets, Vector`

Collections Framework

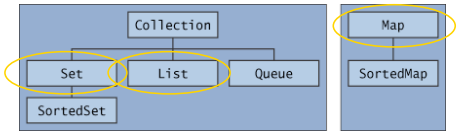
- Unified architecture for representing and manipulating collections
- More than arrays
  - More flexible, functionality, dynamic sizing
- `java.util`

Collections Framework

- **Interfaces**
  - Abstract data types that represent collections
  - Collections can be manipulated *independently* of implementation
- **Implementations**
  - Concrete implementations of the collection interfaces
  - Reusable data structures
- **Algorithms**
  - Methods perform useful computations on collections, e.g., searching and sorting
  - Polymorphic: same method can be used on many different implementations of collection interface
  - Reusable functionality

Core Collection Interfaces

- Encapsulate different types of collections



Generic Collection Interfaces

- Added to 1.5
- Declaration of the Collection interface:
  `public interface Collection<E>...`
  - `<E>` means interface is generic for **e**lement class
- When declare Collection, **specify type** of object contained
  - 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
  `List<Card> hand = new List<Card>();`

Comparable Interface

- Also uses Generics

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

The type it compares

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


Implementation vs. Interface

- Implementation choice affects only performance
- Preferred Style:
  - Choose an implementation
  - Assign collection to variable of corresponding **interface** type or pass collection to method expecting argument of interface type
- Why?
  - Program does not depend on methods in a given implementation
  - Programmer can change implementations
    - Performance concerns or behavioral details

List Interface

- An ordered collection of elements
- Can contain duplicate elements
- Has control over where objects are stored in the list
- `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
- `int size()`
  - Returns the number of elements in the list
- And more! (`contains`, `remove`, `toArray`, …)

Differences from Python

- No shorthand
  - `list[pos]`

List Implementations

- `ArrayList`
  - Resizable array
  - Used most frequently
  - Fast
- `LinkedList`
  - Use if adding elements to beginning of list
  - Use if often delete from middle of list

`cards.Deck.java`