Announcements

Assignment 3 is live! Test forthcoming

Assignment 3 is due Monday at midnight!

Assignment 4 will be due 1/31, and will be short.
Feedback

“Please make sure the unit tests are accurate before releasing the assignment.”

...Sorry. 0:)
Last time...

- Handled exceptions
- Threw our own errors
What we’ve done so far

- Data types
- Primitive data structures
- Object-oriented programming and design
- Syntactical error handling

Now let’s actually do things!
When can you say you *know* Java?

When you know data structures!
Canonical Data structures

Array
Vector
Set
Hash table
Tree
Stack
Queue
Canonical Data structures

Array
Vector
Set
Hash table
Tree
Stack
Queue
Why use data structures?

Because there are certain properties of data structures that help with algorithmic processing. Remember that Java is statically typed! So it makes sense that a data structure needs to know what kind of data it’s processing. Java does this through the generic.
“The term *generic* comes from the idea that we’d like to be able to write general algorithms *that can be broadly reused for many types of objects* rather than having to adapt our code to fit each circumstance. This concept is not new; it is the impetus behind object-oriented programming itself.”
Example: a general list

```
Date d = new Date();
List l = new ArrayList();
l.add(d);
Date temp = (Date) l.get(0); //!
```

How can you ensure that you have a Date object? What do you do if it’s not a Date? (Cry? Crash?)
Generics to the rescue!

From the Java source docs:

```java
public class List<E> {
    ...
    public void add(E element) {
        ...
    }
    public E get(int i) {
        ...
    }
}
```

All we have to tell the data structure is what kind of object!
Now a thought...

Parametrizing the type of the data structure is called *instantiating the type*. Do we always want to instantiate with just any object?
Now a thought...

Parametrizing the type of the data structure is called *instantiating the type*. Do we always want to instantiate with just any object?

Well, depends.
How to bound your generic

```java
public class Bounded<E extends Date>{
    //...
}
```

We don’t exactly know what type `E` will be, but we know it’ll extend `Date`!
Hokay, let’s write a generic class

class Mouse {}
class Bear {}
class Trap<T> {
    T trapped;
    public void snare(T trapped) { this.trapped = trapped; }
    public T release() { return trapped; }
}

Trap<Mouse> trap = new Trap<Mouse>();
trap.snare(new Mouse());
Mouse m = trap.release();
Something Java supports: wildcards

List<?> anyInstantiationOfList =

    new ArrayList<Date>();

anyInstantiationOfList =

    new ArrayList<String>();

And this would be totally fine.

List<? extends Date> = new ArrayList<ModDate>();
So, no surprise: generic functions

class GenericClass< T > { 
  // method using generic class parameter type
  public void T cache( T entry ) { ... } 
}

With bounds:
<T extends Entry & Cacheable > T cache( T entry ) { ... }
So, no surprise: generic functions

class MathUtils {
    public static <T extends Number> T max( T x, T y ) { ... }
}

Bounding a class method!
Back to data structures

Array
Vector
Set
Hash table
Tree
Stack
Queue
Vector<E> vs. ArrayList<E>

A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.

A Vector is thread-safe (but we’ll care about that later.)

Works the same way as an ArrayList!
Back to data structures

Array
Vector
Set
Hash table
Tree
Stack
Queue
Set\langle E\rangle

*Set is an interface, HashSet implements it.* Adding is the same, it just won’t add duplicates.

We’ll code this later.
Back to data structures

Array
Vector
Set
Hash table
Tree
Stack
Queue
The Map<K,V> interface

Map<K,V> is the interface, HashMap is one implementation.
Example

class Contact{ ... }  
HashMap<String, Contact> map =  
    new HashMap<String, Contact>();

map.put("123-456-7890", new Contact(...));
map.put("1-800-MIT-JAVA", new Contact(...));
Contact person = map.get("123-456-7890");
Example

```java
Set<String> set = map.keySet();
Collection<Contact> contacts = map.values();
```
Back to data structures

Array
Vector
Set
Hash table
Tree
Stack
Queue
...does Java have a Tree interface?

Yes, but in the GUI class.

So, let’s write one!
And now for something completely different

Regular expressions!
What’s a regular expression?

A regular expression is just a pattern used to match substrings in a larger text.

Regexs have their own syntax for specifying what it is you want to match.
Why even bother?

- Refactoring purposes
- Being clever
- Useful for email & password validation

‘Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.’ - Jamie Zawinski
Syntax for a regular expression

\d - A digit, 0-9
\D - Anything except a digit
\Q ... \E - a literal match for anything in between
. - Any character
\s - literal-space character
\S - any character except whitespace
Syntax for a regular expression

\t - A tab
\r - A return
\n - new line
\w - A-Z, a-z, 0-9, the underscore
\W - Anything except those things
Custom character classes

“I want at least ONE of either a or b.”
[ab]

“I want a letter, I don’t care about the case.”
[A-za-z]

“I want anything EXCEPT lowercase a and b.”
[^ab]

“I want the overlap of a-p and l-p.”
[a-p&&[l-p]]
A rose is a rose is a rose...

If we tried “[Aa] rose”, it would match three times. But say we wanted to match only at the front or back.

^[Aa] rose - would match at the beginning

[Aa] rose$ - matches at the end
Boundaries

\b - Match a word boundary
\B - Match a nonword boundary

Example:
\brose would match rose and rosemary, but not primrose.
Multiplicity

* - Zero or more
+ - One or more
? - Optionally zero or more
{x,} - At least x or more
{x} - Exactly x many
{x} - Exactly x many
{x,y} - At least x many, no more than y many
Multiplicity examples

0*\d // Match a digit with any number of leading zeros
0*\d+ // Match one or more digits with any number of leading zeros
\d\d/?\d\d // Match four digits, a slash in between is allowed
Groupings

(...) - We’ve made a subexpression!

(text)+ - I want to see “text” one or more times.

\w+@\w+(\.|\w)+ - The regular expression for an email
Logical operators

\( a \lor b = \text{either } a \text{ or } b. \)

Email domain matching example:

\( \w+@[\w.]\*(\text{net}|\text{edu}|\text{gov}) \)
import java.util.regex.*;

boolean match = Pattern.matches("\\d+\\.\\d+f?", myText);

// The regex for float numbers
The Matcher class

```java
import java.util.regex.*;

//...
String text = "A horse is a horse, of course of course...";
String pattern = "horse|course";
Matcher matcher = Pattern.compile(pattern).matcher(text);

while (matcher.find()) {
    System.out.println("Matched: "+matcher.group() +" at position "+matcher.start());
}
```
Output

Matched: 'horse' at position 2
Matched: 'horse' at position 13
Matched: 'course' at position 23
Matched: 'course' at position 33
Matcher instance method highlights

matches() - Does my entire string match this regex at all?
lookingAt() - “Attempts to match the input sequence, starting at the beginning of the region, against the pattern.”
find() - Try to match the next subsequence after the previous one.
replaceAll(String replacement) - Replace every match found in the string with replacement.

Matcher basically acts as a scanner through a string!
Splitting a String

We can use regexes to do this!
Splitting a String

String text = "Foo, bar ,   blah";
String [] fields = text.split( "\s*,\s*" );
//fields = {"Foo", "bar", "blah"};

// Text.split is the same as
// Pattern.compile(pattern).split(text);
Replacement

String text = "Richard Nixon's social security number is: 567-68-0515.";
Matcher matcher =
    Pattern.compile("\\d\\d\\d-\\d\\d-\\d\\d\\d-\\d\\d\\d\\d\\d").matcher( text );
String output =
    matcher.replaceAll("XXX-XX-XXXX");
Sanity check

How could we improve \d\d\d-\d\d\d-\d\d\d\d\d?
Sanity check

How could we improve `\d\d\d-\d\d-\d\d\d\d`?

`\d{3}-\d{2}-\d{4}`!