```
try
{
    Assert(Life.Real);
    Assert(Life.Fantasy);
}
catch(LandSlideException ex)
{
     #region Reality
     while(true)
     {
         character.Eyes.ForEach(eye => eye.Open().Orient(Direction.Sky).See(););
         self.Wealth = null;
         self.Sex = Sex.Male;

         if(self.ComeDifficulty == Difficulty.Easy && self.GoDifficulty ==
    Difficulty.Easy && self.High < 0.1 && self.Low < 0.1)
         {
             self.Sympathies.Clear();

             switch(wind.Direction)
             {
                 case Direction.North:
                 case Direction.East:
                 case Direction.South:
                 case Direction.West:
                 default:
                 piano.Play();
                 break;
             }
         }
     }
     #endregion
}
```

"Bohemian Rhapsody"

Section 6:
HW6 and Interfaces

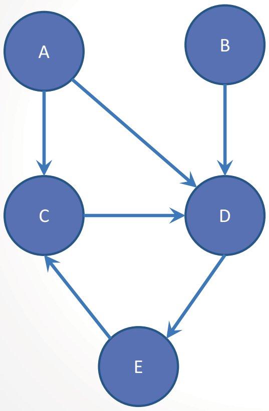
Slides by Alex Mariakakis

with material from Krysta Yousoufian,
Mike Ernst, Kellen Donohue

Agenda

- BFS
- Interfaces
- Parsing Marvel Data

Graphs



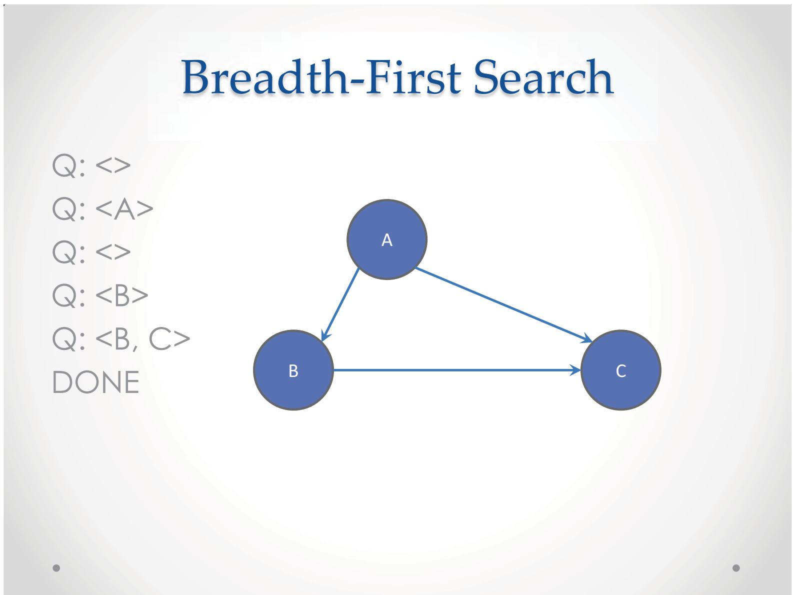
Can I reach B from A?

Breadth-First Search (BFS)

- Often used for discovering connectivity
- Calculates the shortest path if and only if all edges have same positive or no weight
- Depth-first search (DFS) is commonly mentioned with BFS
  - BFS looks "wide", DFS looks "deep"
  - Can also be used for discovery, but not the shortest path

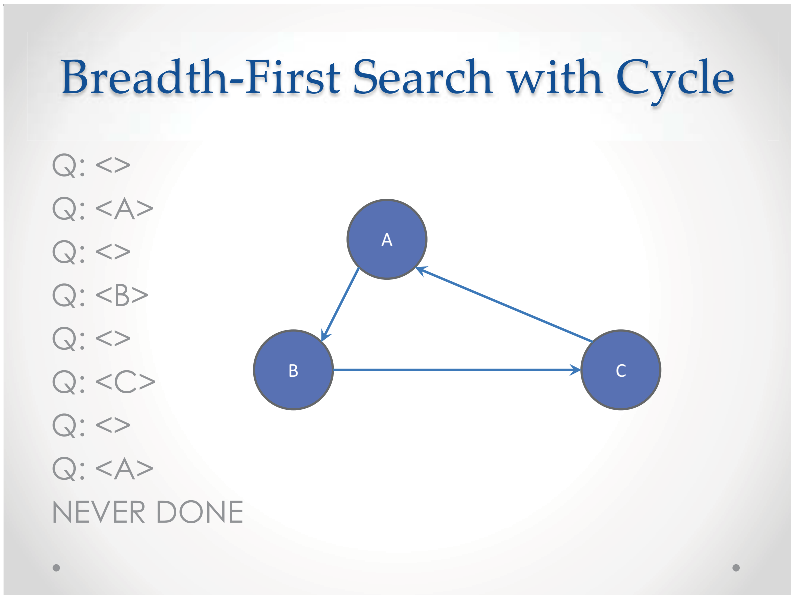
BFS Pseudocode

```
public boolean find(Node start, Node end) {
        put start node in a queue
        while (queue is not empty) {
                pop node N off queue
                if (N is goal)
                        return true;
                else {
                        for each node O that is child of N
                                push O onto queue
                }
        }
        return false;
}
```



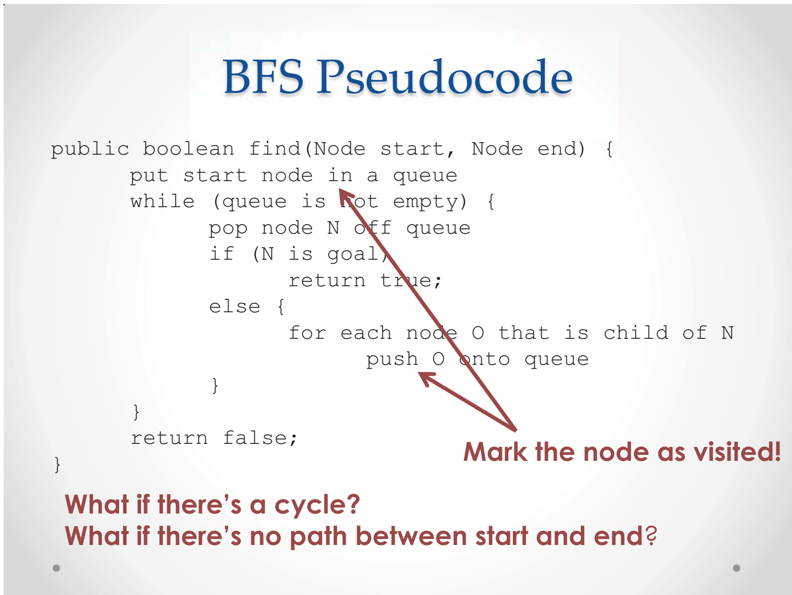
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <B>
Q: <B, C>
DONE



Breadth-First Search with Cycle

Q: <>
Q: <A>
Q: <>
Q: <B>
Q: <>
Q: <C>
Q: <>
Q: <A>
NEVER DONE

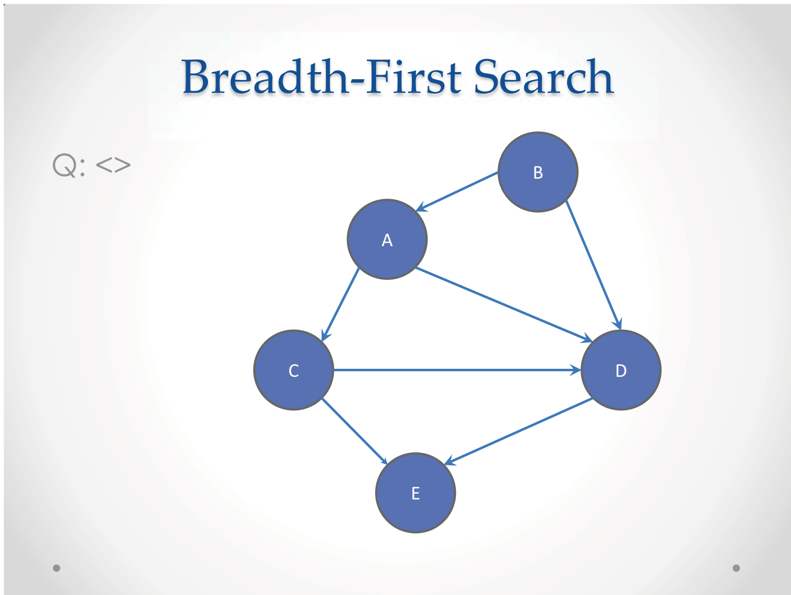


BFS Pseudocode

```
public boolean find(Node start, Node end) {
      put start node in a queue
      while (queue is not empty) {
            pop node N off queue
            if (N is goal)
                  return true;
            else {
                  for each node O that is child of N
                        push O onto queue
            }
      }
      return false;
}
```

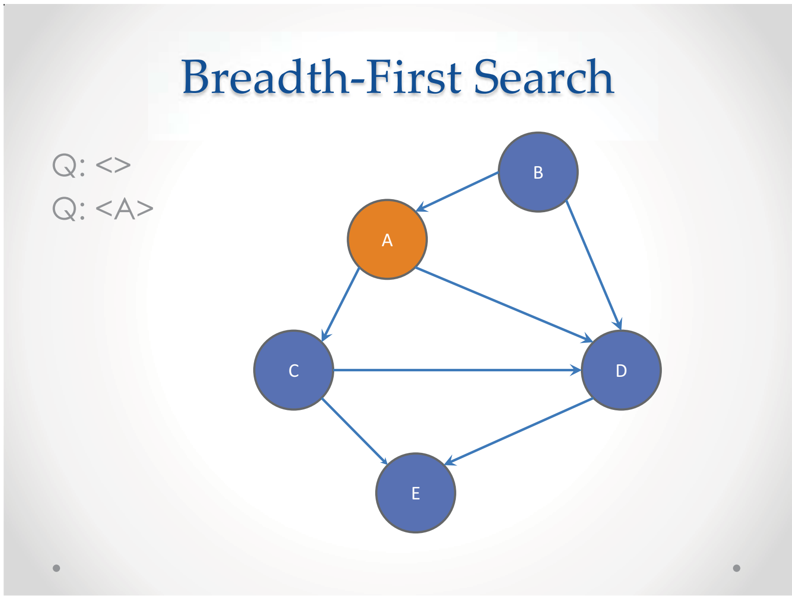
Mark the node as visited!

What if there's a cycle?
What if there's no path between start and end?



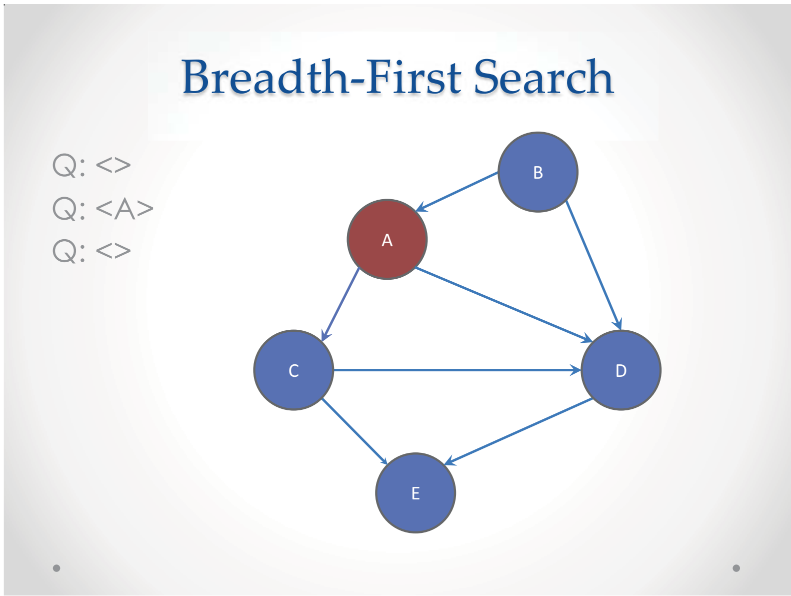
Breadth-First Search

Q: <>



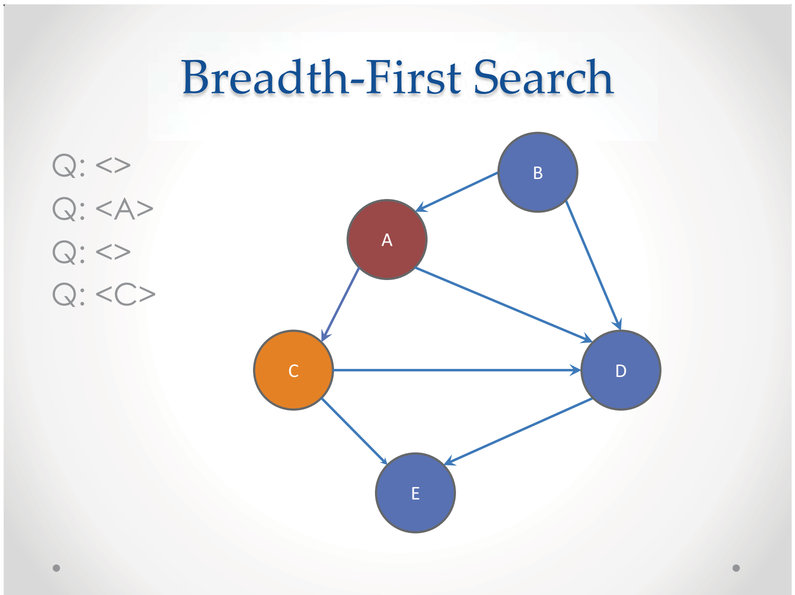
Breadth-First Search

Q: <>
Q: <A>



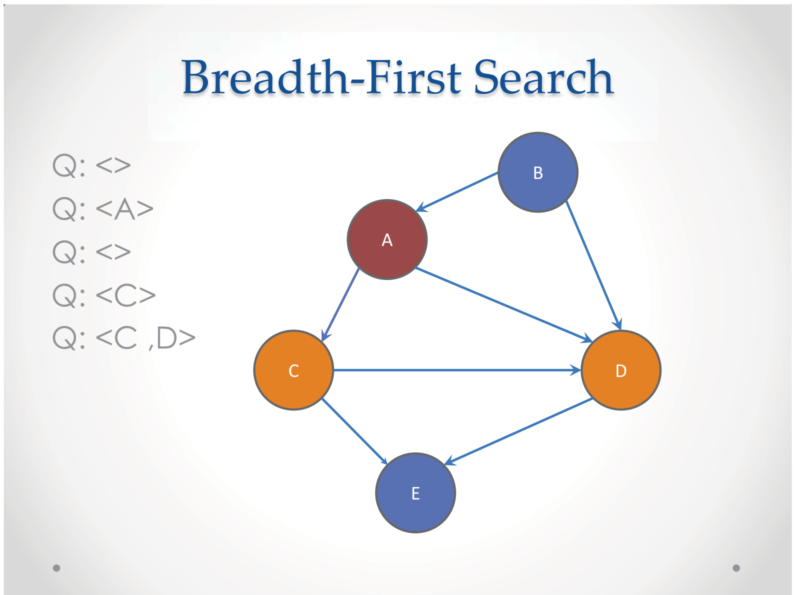
Breadth-First Search

Q: <>
Q: <A>
Q: <>



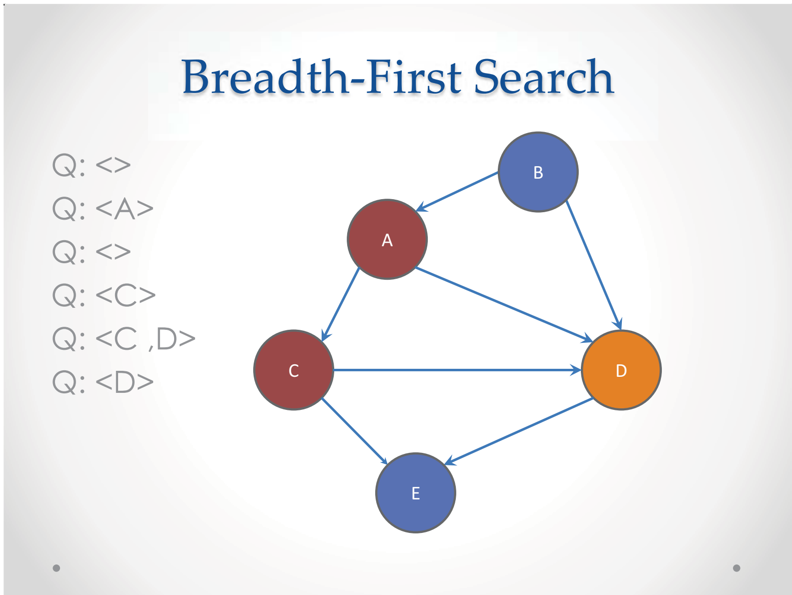
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>



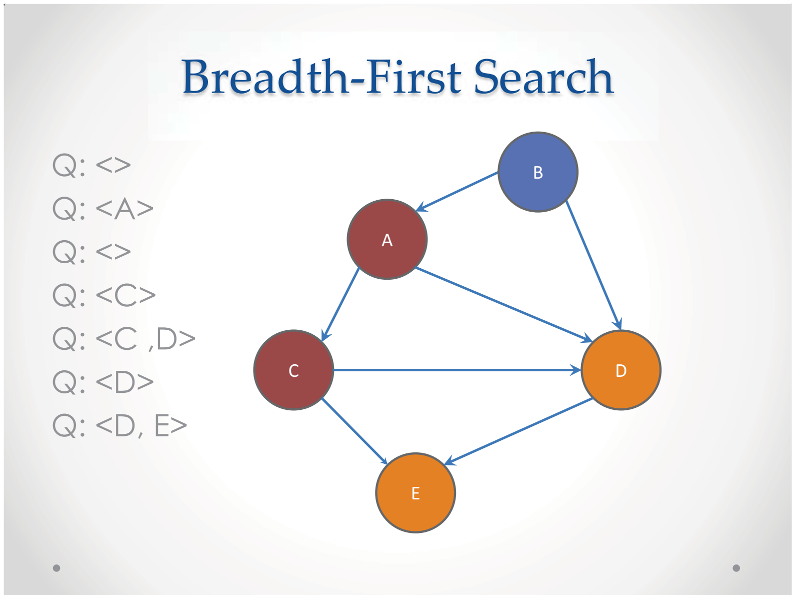
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C ,D>



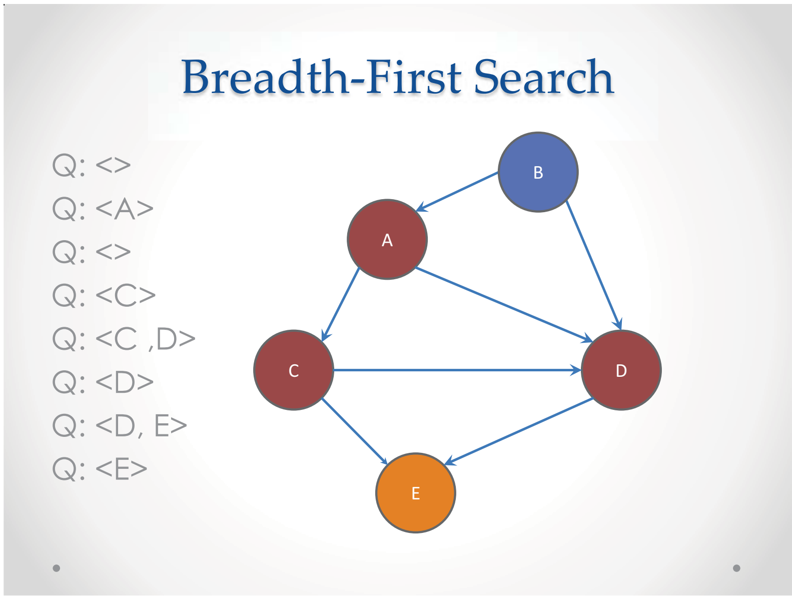
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C ,D>
Q: <D>



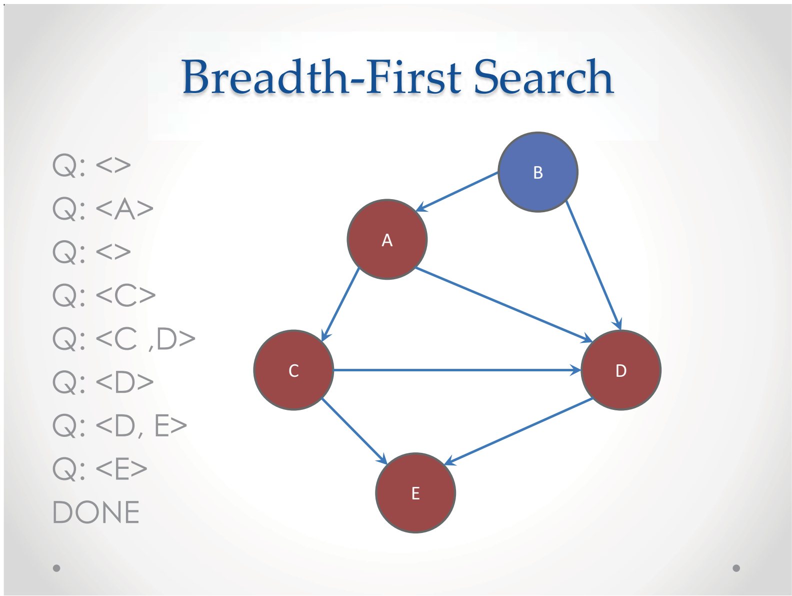
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C ,D>
Q: <D>
Q: <D, E>



Breadth-First Search

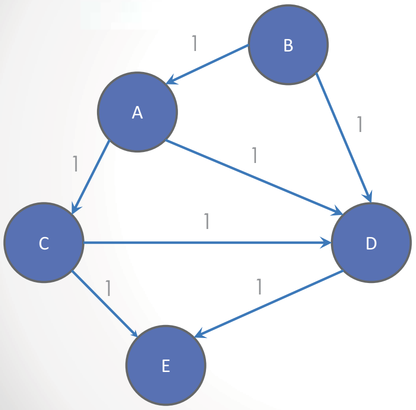
Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C ,D>
Q: <D>
Q: <D, E>
Q: <E>



Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C ,D>
Q: <D>
Q: <D, E>
Q: <E>
DONE

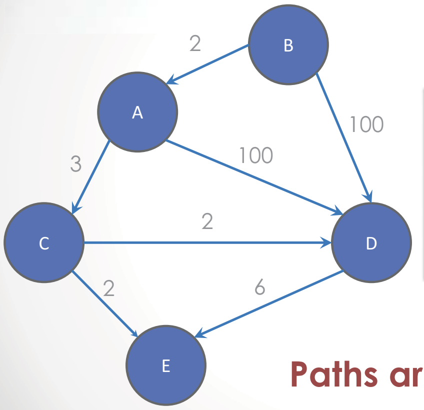
Shortest Paths with BFS



From Node B

Destination	Path	Cost
A	<B,A>	1
B	<B>	0
C	<B,A,C>	2
D	<B,D>	1
E	<B,D,E>	2

Shortest Paths with Weights



From Node B

Destination	Path	Cost
A	<B,A>	2
B	<B>	0
C	<B,A,C>	5
D	<B,A,C,D>	7
E	<B,A,C,E>	7

Paths are not the same!

Classes, Interfaces, and Types

- The fundamental unit of programming in Java is a class
- Classes can extend other classes and implement interfaces
- Interfaces can extend other interfaces

Classes, Objects, and Java

- Everything is an instance of a class
  - Defines data and methods
- Every class extends exactly one other class
  - Object if no explicit superclass
  - Inherits superclass fields
- Every class also defines a type
  - Foo defines type Foo
  - Foo inherits all inherited types
- Java classes contain both specification and implementation!

Interfaces

- Pure type declaration

```
public interface Comparable {
     int compareTo(Object other);
}
```

- Can contain:
  - Method specifications (implicitly `public abstract`)
  - Named constants (implicitly `public final static`)
- Does not contain implementation
- Cannot create instances of interfaces

Implementing Interfaces

- A class can implement one or more interfaces

```
class Kitten implements Pettable, Huggable
```

- The implementing class and its instances have the interface type(s) as well as the class type(s)
- The class must provide or inherit an implementation of all methods defined by the interface(s)
  - Not true for abstract classes

Using Interface Types

- An interface defines a type, so we can declare variables and parameters of that type
- A variable with an interface type can refer to an object of any class implementing that type

```
List<String> x = new ArrayList<String>();
void sort(List myList) {…}
```

Guidelines for Interfaces

- Provide interfaces for significant types and abstractions
- Write code using interface types like Map instead of HashMap and TreeMap wherever possible
  - Allows code to work with different implementations later on
- Both interfaces and classes are appropriate in various circumstances

Demo

Parsing the Marvel data