Linked Lists, Part 1

Please take a handout &
Sit in row H or forward

Look at all the things we've done!

Concepts / skills:

functional programming

imperative programming

code analysis
summations • recurrence relations • Big O

sequential data structures
linked lists • arrays

higher-order functions
map • fold • filter • lambda • functions as data

object-oriented programming
classes • object • methods • fields • this • static

memory model
"box and arrow" diagrams

Professional practices:

write tests (early)!

write documentation (early)!
inline comments • high-level comments • JavaDoc

reuse as much code as possible

use a style guide
always use this • call existing methods if possible

This week: implementing linked lists

We'll **revisit**:

~~functional programming~~

imperative programming

code analysis

summations • ~~recurrence relations~~ • Big O

sequential data structures

linked lists • arrays

~~higher-order functions~~

~~map • fold • filter • lambda • functions as data~~

object-oriented programming

classes • object • methods • fields • this • static

memory model

"box and arrow" diagrams

We'll **introduce**:

how to make new data structures

packages

private, inner classes

How to create a new data structure

Three steps to think about

Operations: what can the data structure do?

- The operations are the **interface**: the public methods & fields.
- The operations can be specified in a Java `interface`,
  if there might be multiple ways to implement the same interface.
- An operation's cost is sometimes an unofficial part of the interface.

A Point's operations

purpose	signature	cost
read the X-coordinate	`int getX()`	O(1)
read the Y-coordinate	`int getY()`	O(1)
translate the point by a specified amount in the X & Y direction	`void move(int deltaX, int deltaY)`	O(1)

and, because we're implementing our Point in Java:

purpose	signature	cost
at least one constructor	*depends on class name*	O(1)
represent instance as `String`	`String toString()`	O(1)
compare to another `Object`	`boolean equals(Object obj)`	O(1)
compute a hash code *so, e.g., instances can be placed in a dictionary*	`int hashCode()`	O(1)

How to create a new data structure

Three steps to think about

Operations: what can the data structure do?

- The operations are the **interface**: the public methods & fields.
- The operations can be specified in a Java `interface`,
  if there might be multiple ways to implement the same interface.
- An operation's cost is sometimes an unofficial part of the interface.

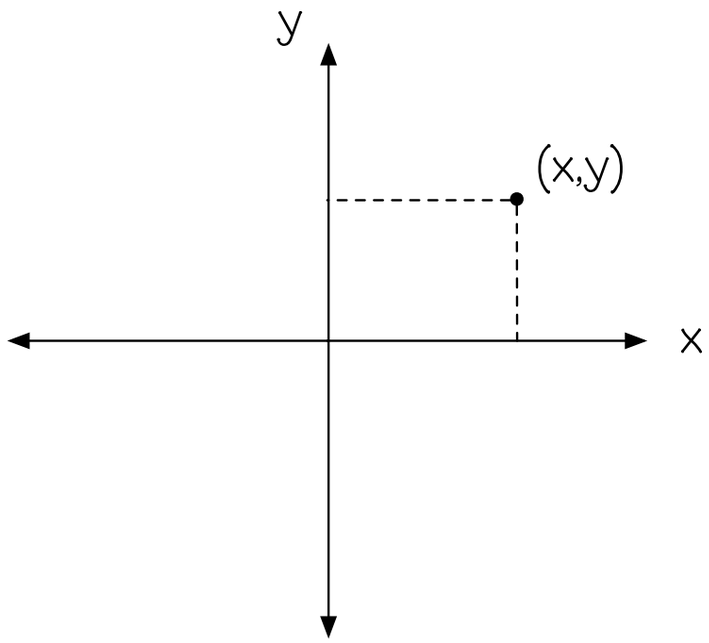
Representation: how will the structure store its data?

- The representation is part of the **implementation**: the private fields.
- The fields are themselves data structures!

A Point's representation

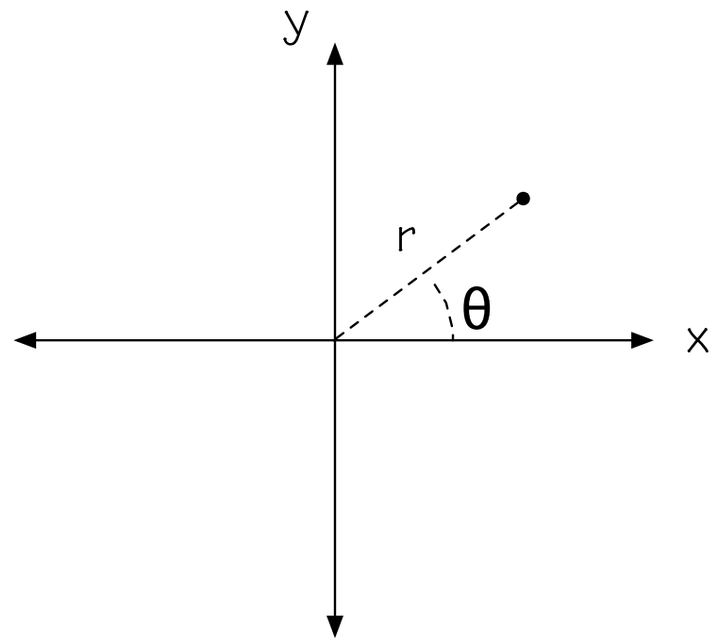
Drawing pictures is an excellent way to describe the representation

Cartesian (rectangular)



purpose	signature
horizontal component	`int x`
vertical component	`int y`

Polar



purpose	signature
magnitude	`int distance`
angle	`int angle`

the fields could be doubles, too

How to create a new data structure

Three steps to think about

Operations: what can the data structure do?

- The operations are the **interface**: the public methods & fields.
- The operations can be specified in a Java `interface`,
  if there might be multiple ways to implement the same interface.
- An operation's cost is sometimes an unofficial part of the interface.

Representation: how will the structure store its data?

- The representation is part of the **implementation**: the private fields.
- The fields are themselves data structures!

Implementation: write the operations

- the bodies of the public methods.
- use the representation
- add private helper methods as needed
- re-use as much code as possible

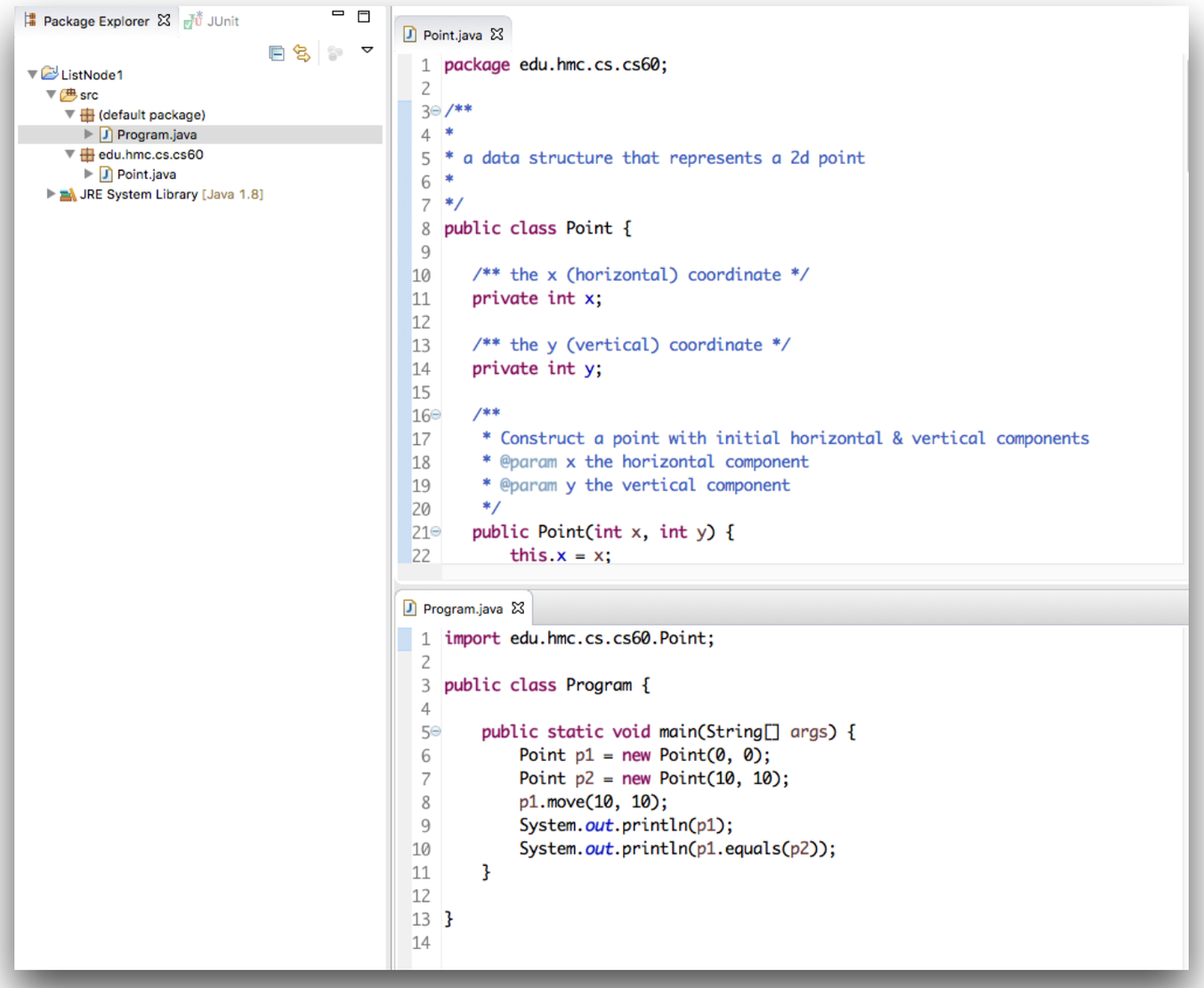
A Point's implementation

Using Cartesian (rectangular) coordinates

Good programming practice

Put your code inside a package.

That way, your class has a **unique** name.



A Point's implementation

Using Cartesian (rectangular) coordinates

1. Package declaration

```
package edu.hmc.cs.cs60;
```

2. Class declaration

```
public class Point {
   …
}
```

3. Method declarations

```
/**
 * Translate a point to a different location
 * @param deltaX the horizontal distance to translate
 * @param deltaY the vertical distance to translate
 */
public void move(int deltaX, int deltaY) {
    // TODO: implement me!
}
```

4. Tests!

```
public class PointTest {
   @Test
    …
}
```

5. Field declarations

```
/** the x (horizontal) coordinate */
private int x;

/** the y (vertical) coordinate */
private int y;
```

6. Method implementations

```
public void move(int deltaX, int deltaY) {
   this.setX(this.getX() + deltaX);
   this.setY(this.getY() + deltaY);
}
```

upload.wikimedia.org/wikipedia/commons/8/8c/Key_break.jpg

Let's implement a linked list (of ints)!

Why?

A linked list's operations

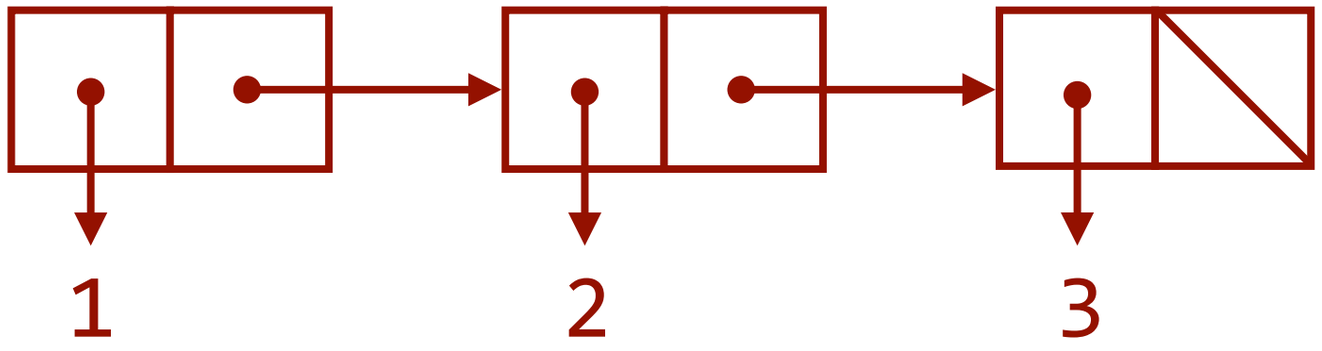
purpose	signature	cost
test for emptiness	`boolean isEmpty()`	O(1)
the length of the list	`int size()`	O(1)
prepend an element (just like cons)	`void addToFront(int i)`	O(1)
true if *i* is in the list	`boolean contains(int i)`	O(n)
returns the i^{th} element of the list	`int get(int i)`	O(n)

and, because we're implementing our linked list in Java:

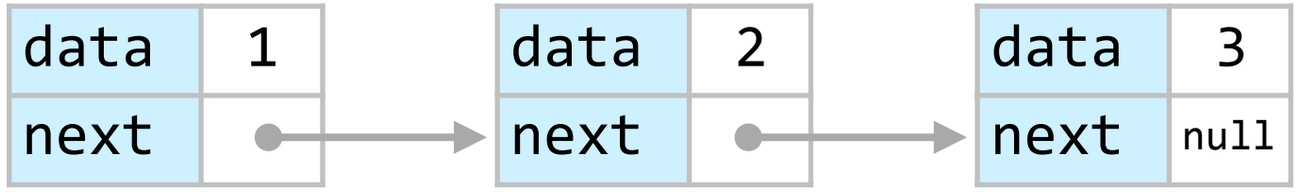
purpose	signature	cost
at least one constructor	`List()`	O(1)
represent instance as `String`	`String toString()`	O(n)
compare to another `Object`	`boolean equals(Object obj)`	O(n)
compute a hash code *so, e.g., instances can be placed in a dictionary*	`int hashCode()`	O(n)

A linked list's representation

Drawing pictures is an excellent way to design the representation



a linked list in Racket

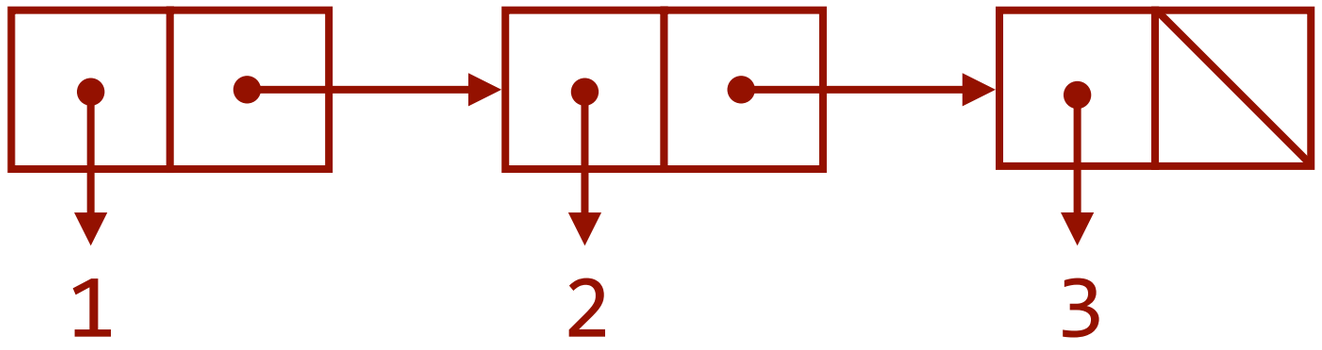


a linked list in Java

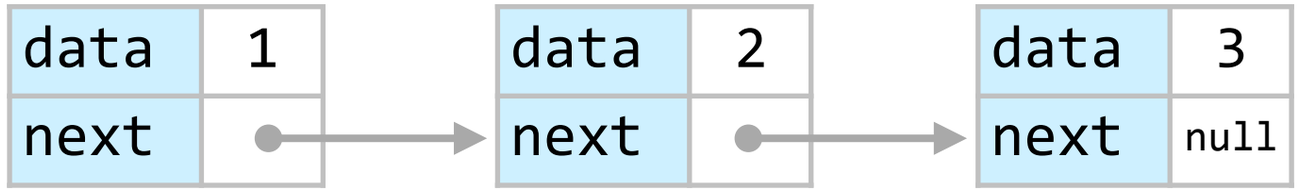
purpose	signature
the list element	`??? data`
a reference to the rest of the list	`??? next`

A linked list's representation

Drawing pictures is an excellent way to design the representation



a linked list in Racket



a linked list in Java

purpose	signature
the list element	`int data`
a reference to the rest of the list	`List next`

How would you represent the **empty list**?