CSE 331
Software Design & Implementation

Kevin Zatloukal

Spring 2020

Abstract Data Types (ADTs)

Procedural and data abstractions

Procedural abstraction:

- abstract from implementation details of *procedures* (methods)
- specification is the abstraction
- satisfy the specification with an implementation

Data abstraction:

- abstract from details of *data representation*
- also a specification mechanism
- way of thinking about programs and design

Abstract Data Type (ADT)

- invented by Barbara Liskov in the 1970s
- one of the fundamental ideas of computer science

Why we need Data Abstractions (ADTs)

Manipulating and presenting data is pervasive

- choosing how to organize that data is key design problem
- inventing and describing algorithms is less common

ADTs give us the freedom to **change** data structures later on

- data structure details are hidden from the clients

Pro tip: often best to start by designing data

- first, what **operations** will be permitted on the data (for clients)
- next, decide how data be organized (data structures)
- lastly, write the code

Specifying an ADT

Immutable

1. **overview**
2. **abstract state**
3. **creators**
4. **observers**
5. **producers**
6. ~~**mutators**~~

Mutable

1. **overview**
2. **abstract state**
3. **creators**
4. **observers**
5. **producers (rare)**
6. **mutators**

- Should have no information about the implementation
  - (latter called the "concrete representation")
  - leave ourselves free to change it later
- A collection of procedural abstractions — not procedures

Concept of 2D point, as an ADT

```
class Point {
  // A 2D point exists in the plane, ...
  public float x();
  public float y();
  public float r();
  public float theta();

  // ... can be created, ...
  public Point(); // new point at (0,0)
  public Point centroid(Set<Point> points);

  // ... can be moved, ...
  public void translate(float delta_x,
                        float delta_y);
  public void scaleAndRotate(float delta_r,
                             float delta_theta);
}
```

Observers / Getters (x, y, r, theta)

Creators / Producers (Point, centroid)

Mutators (translate, scaleAndRotate)

Poly, an immutable datatype: overview

```
/**
 * A Poly is an immutable polynomial with
 * integer coefficients.  A typical Poly is
 *           c0 + c1x + c2x^2 + ...
 */
class Poly {
```

Overview: describes what the object means / represents

- state if immutable (default not)
- define <u>abstract states</u> for use in operation specifications
  - difficult and vital!
  - appeal to math if appropriate
- give an example (reuse it in operation definitions)

Poly: creators

```
// effects: makes a new Poly = 0
public Poly()

// effects: makes a new Poly = cxⁿ
// throws: NegExponent if n < 0
public Poly(int c, int n)
```

Creators: creates a new object

- no pre-state: only **`effects`**, no **`modifies`**
- overloading: distinguish procedures of same name by parameters
  - use with care (see Effective Java)
  - will see alternative design patterns later on

Poly: observers

```
// returns: the degree of this polynomial,
//   i.e., the largest exponent with a
//   non-zero coefficient.
//   Returns 0 if this = 0.
public int degree()
```

"this" means the abstract state

```
// returns: the coefficient of the term
//   of this polynomial whose exponent is d
// throws: NegExponent if d < 0
public int coeff(int d)
```

Observers: retrieves information about the abstract state

- **never** modify the abstract state

Poly: producers

```
// returns: this + q
public Poly add(Poly q)

// returns: this * q
public Poly mul(Poly q)

// returns: -this
public Poly negate()
```

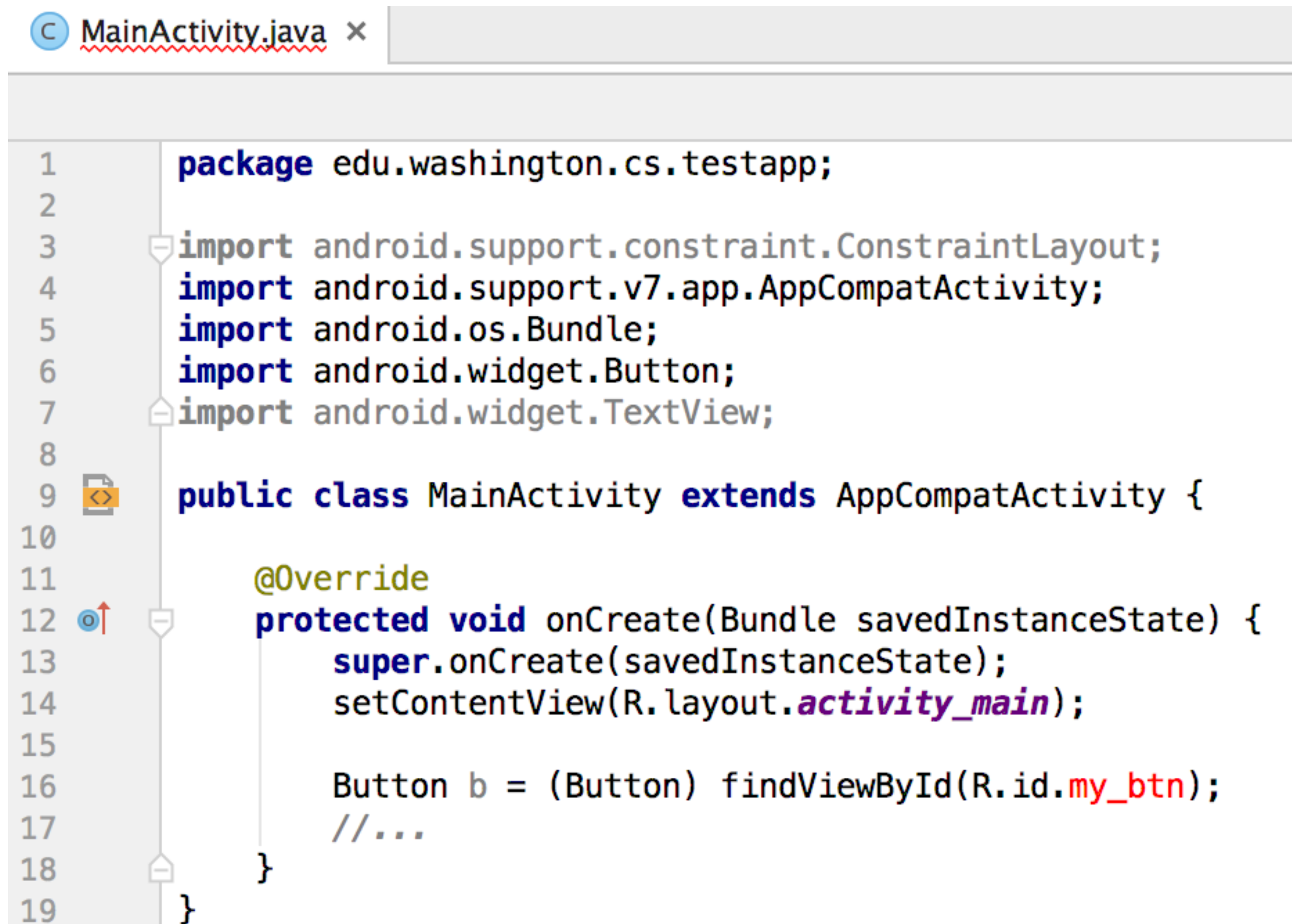
Producers: creates other objects of the same type

- **never** modify the abstract value of existing objects

Another Example

Example: Text File

Use case is writing an editor for an IDE:



Example: Text File

Overview: telling users how to think about what this is

Option 1: list of characters & colors
Option 2: list of lines, each of which is a...
list of characters & colors

Both will probably require a method to take (line, col) to character

Key difference:

- Option 1 suggests you can remove, e.g., chars 100–200, which may span multiple lines
- That is not natural in Option 2

(Option 1 makes more sense for Microsoft Word.)

Example: Text File

Will use a list of lines.
What is each **line**?

Option 1: pair (list of characters, list of colors)
Option 2: list of pairs (character, color)
Option 3: list of pairs (list of characters, color)

Option 1 must make clear that the lists are same length

Key differences:

- Option 1 & 2 should let you insert (char, color) at given column
- Option 3 should let change the color of a keyword, which is a single (chars, color), in one operation

Example: Text File

```
// Overview: Represents a text file, which is a list of
// lines of text. Each line of text is a list of
// (character, color) pairs.
//
// Example: [[(“a”, black), (“b”, red)], [(“c”, green)]]
// is the text:
//   ab
//   c
// (on two lines), where a is black, b is red, & c is green
public class TextFile {

  // ...

}
```

Building Blocks of Abstract States

Some useful “math” concepts for describing states abstractly

- numbers
- characters
- lists
- tuples (fixed length)
- objects
  - parts are named, not numbered (as in tuples)
  - e.g. {chars: “protected”, color: 3}