Warmup

["hip","hip"]

=

Hip Hip Array!

Section 3:
HW4, ADTs, and more

Slides by Alex Mariakakis

with material from Krysta Yousoufian,
Mike Ernst, Kellen Donohue

Agenda

- HW4 setup
- Abstract data types (ADTs)
- Method specifications

HW#4 DEMO

Polynomial Addition

$$(5x^4 + 4x^3 - x^2 + 5) + (3x^5 - 2x^3 + x - 5)$$

$$\begin{array}{r} 5x^4 + 4x^3 - x^2 + 0x + 5 \\ + \; 3x^5 + 0x^4 - 2x^3 + 0x^2 + x - 5 \\ \hline 3x^5 + 5x^4 - 2x^3 - x^2 + x + 0 \end{array}$$

Polynomial Multiplication

$(4x^3 - x^2 + 5) * (x - 5)$

$$
\begin{array}{rr}
 & 4x^3 - x^2 + 5 \\
* & x - 5 \\
\hline
 & -20x^3 + 5x^2 \quad - 25 \\
+ & 4x^4 \quad -x^3 \quad + 5x \\
\hline
 & \mathbf{4x^4 - 21x^3 + 5x^2 + 5x - 25}
\end{array}
$$

Polynomial Division

$(5x^6 + 4x^4 - x^3 + 5) \;/\; (x^3 - 2x - 5)$

$x^3 - 2x - 5 \;\overline{)\; 5x^6 + 4x^4 - x^3 + 5}$

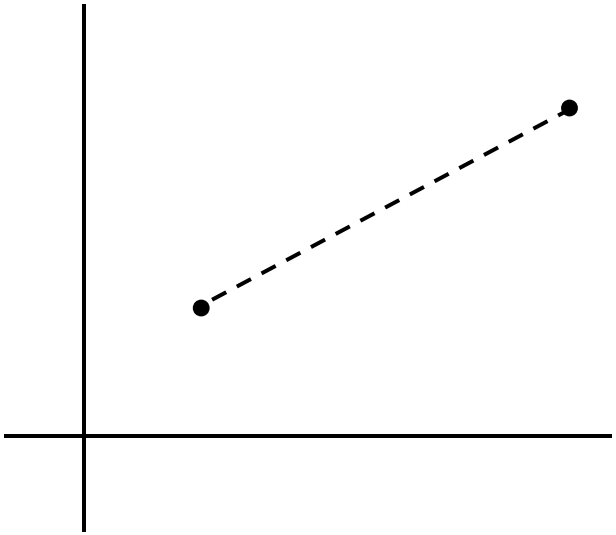
Polynomial Division

$$
\begin{array}{r|l}
 & 5\ \ 0\ \ 14\ \ 24 \\
\hline
1\ \ 0\ \ -2\ \ -5 & 5\ \ 0\ \ 4\ \ -1\ \ 0\ \ 0\ \ 5 \\
 & -\ 5\ \ 0\ \ -10\ \ -25 \\
\hline
 & \ \ \ 0\ \ 14\ \ 24\ \ 0 \\
 & -\ 0\ \ \ 0\ \ \ 0\ \ \ 0 \\
\hline
 & \ \ \ \ 14\ \ 24\ \ 0\ \ 0 \\
 & -\ 14\ \ 0\ \ -28\ \ -70 \\
\hline
 & \ \ \ \ \ \ 24\ \ 28\ \ 70\ \ 5 \\
 & -\ \ 24\ \ \ 0\ \ -48\ \ -120 \\
\hline
 & \ \ \ \ \ \ 0\ \ 28\ \ 118\ \ 125
\end{array}
$$

$$
\begin{array}{r}
5x^3 + 14x + 24 \\
+\ \ 28x^2 + 118x + 125 \\
\hline
x^3 - 2x - 5
\end{array}
$$

ADT Example: Line

Suppose we want to make a `Line` class that represents lines on the Cartesian plane



See http://courses.cs.washington.edu/courses/cse331/13au/conceptual-info/specifications.html for more

Definitions

- **Abstract Value**: what an instance of a class is supposed to represent
  - `Line` represents a given line
- **Abstract State**: the information that defines the abstract value
  - Each line has a start point and an end point
- **Abstract Invariant:** the conditions that must remain true over the abstract state for all instances
  - Start point and end point must be distinct

Definitions (cont.)

- **Specification Fields**: describes components of the abstract state of a class
  - `Line` has specification fields `startPoint`, `endPoint`
- **Derived Specification Fields**: information that can be derived from specification fields but useful to have
  - `length = sqrt((x1-x2)^2 + (y1-y2)^2)`

ADT Example: Line

```
/**
 * This class represents the mathematical concept of a line segment.
 *
 * Specification fields:
 *  @specfield start-point : point  // The starting point of the line.
 *  @specfield end-point   : point  // The ending point of the line.
 *
 * Derived specification fields:
 *  @derivedfield length : real     // The length of the line.
 *
 * Abstract Invariant:
 *  A line's start-point must be different from its end-point.
 */
public class Line {
…
}
```

ADT Example: Line

```
/**
 * This class represents the mathematical concept of a line segment.
 *
 * Specification fields:
 *  @specfield start-point : point  // The starting point of the line.
 *  @specfield end-point   : point  // The ending point of the line.
 *
 * Derived specification fields:
 *  @derivedfield length : real     // The length of the line.
 *
 * Abstract Invariant:
 *  A line's start-point must be different from its end-point.
 */
public class Line {
…
}
```

Abstract Value

ADT Example: Line

```
/**
 * This class represents the mathematical concept of a line segment.
 *
 * Specification fields:
 *  @specfield start-point : point  // The starting point of the line.
 *  @specfield end-point   : point  // The ending point of the line.
 *
 * Derived specification fields:
 *  @derivedfield length : real     // The length of the line.
 *
 * Abstract Invariant:
 *  A line's start-point must be different from its end-point.
 */
public class Line {
…
}
```

Abstract State

ADT Example: Line

```
/**
 * This class represents the mathematical concept of a line segment.
 *
 * Specification fields:
 *  @specfield start-point : point  // The starting point of the line.
 *  @specfield end-point   : point  // The ending point of the line.
 *
 * Derived specification fields:
 *  @derivedfield length : real     // The length of the line.
 *
 * Abstract Invariant:
 *  A line's start-point must be different from its end-point.
 */
public class Line {
…
}
```

Abstract Invariant

ADT Example: Line

```
/**
 * This class represents the mathematical concept of a line segment.
 *
 * Specification fields:
 *  @specfield start-point : point  // The starting point of the line.
 *  @specfield end-point   : point  // The ending point of the line.
 *
 * Derived specification fields:
 *  @derivedfield length : real     // The length of the line.
 *
 * Abstract Invariant:
 *  A line's start-point must be different from its end-point.
 */
public class Line {
…
}
```

Specification Fields

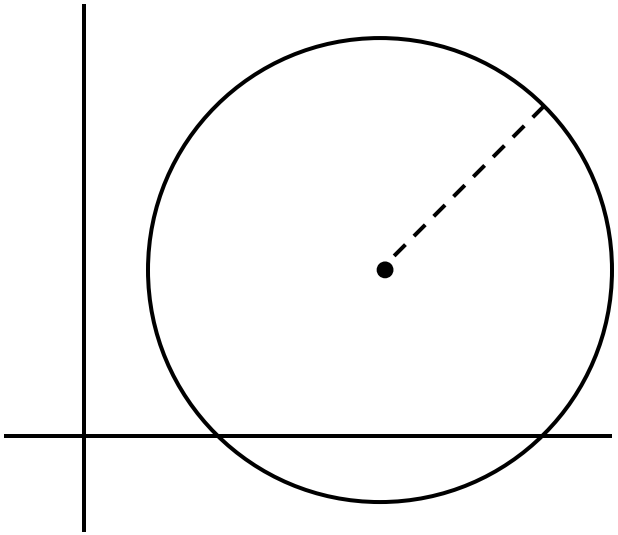
ADT Example: Line

```
/**
 * This class represents the mathematical concept of a line segment.
 *
 * Specification fields:
 *  @specfield start-point : point  // The starting point of the line.
 *  @specfield end-point   : point  // The ending point of the line.
 *
 * Derived specification fields:
 *  @derivedfield length : real     // The length of the line.
 *
 * Abstract Invariant:
 *  A line's start-point must be different from its end-point.
 */
public class Line {
…
}
```

Derived Fields

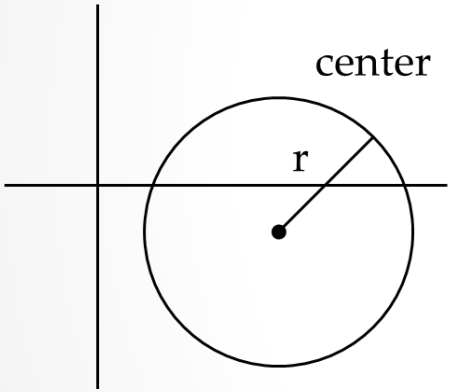
ADT Example: Circle

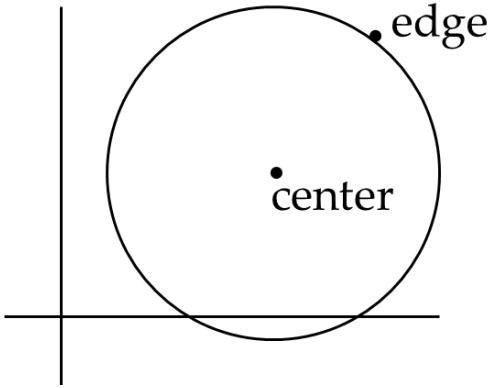
Suppose we want to make a `Circle` class that represents circles on the Cartesian plane

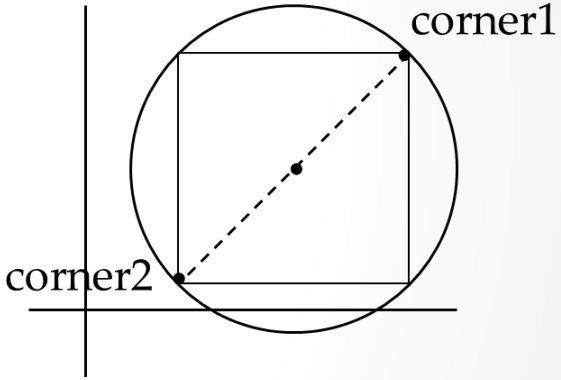


ADT Example: Circle

- **Abstract Value**:
  - `Circle` represents a given circle
- **Abstract State**:







- **Abstract Invariant**
  - Option #1: r > 0, center must exist
  - Option #2: center and edge must be distinct
  - Option #3: corner1 and corner2 must be distinct

ADT Example: Circle

- **Specification Fields**:
  - Option #1: `r` and `center`
  - Option #2: `center` and `edgePoint`
  - Option #3: `corner1` and `corner2`
- **Derived Specification Fields**:
  - Circumference
  - Diameter
  - Area
  - …

Abstraction

- Abstract values, state, and invariants specify the behavior of classes and methods
  - What should my class do?
- We have not implemented any of these ADTs yet
  - Implementation should not affect abstract state
  - As long as `Circle` represents the circle we are interested in, nobody cares how it is implemented

Abstract vs. Concrete

- We'll talk later about **representation invariants**, which specify how the abstract invariant is implemented
- We'll also discuss how **abstraction functions** map the concrete representation of an ADT to the abstract value

Javadoc Documentation

- Tool made by Oracle for API documentation
- We've already seen Javadoc for external class specification
- Method specifications will describe method behavior in terms of preconditions and postconditions

Javadoc Method Tags

- **`@requires`**: the statements that must be met by the method's caller
- **`@return`**: the value returned by the method, if any
- **`@throws`**: the exceptions that may be raised, and under which conditions
- **`@modifies`**: the variables that *may* change because of the method
- **`@effects`**: the side effects of the method

Javadoc Method Tags

- If **`@requires`** is not met, anything can happen
  - False implies everything
- The conditions for **`@throws`** must be a subset of the precondition
  - Ex: If a method **`@requires`** x > 0, **`@throws`** should not say anything about x < 0
- **`@modifies`** lists *what* may change, while **`@effects`** indicates *how* they change
  - If a specification field is listed in the **`@modifies`** clause but not in the **`@effects`** clause, it may take on any value (provided that it follows the abstract invariant)
  - If you mention a field in **`@modifies`**, you should try to specify what happens in **`@effects`**