Specifications

Krysta Yousoufian

CSE 331 Section

April 12, 2012

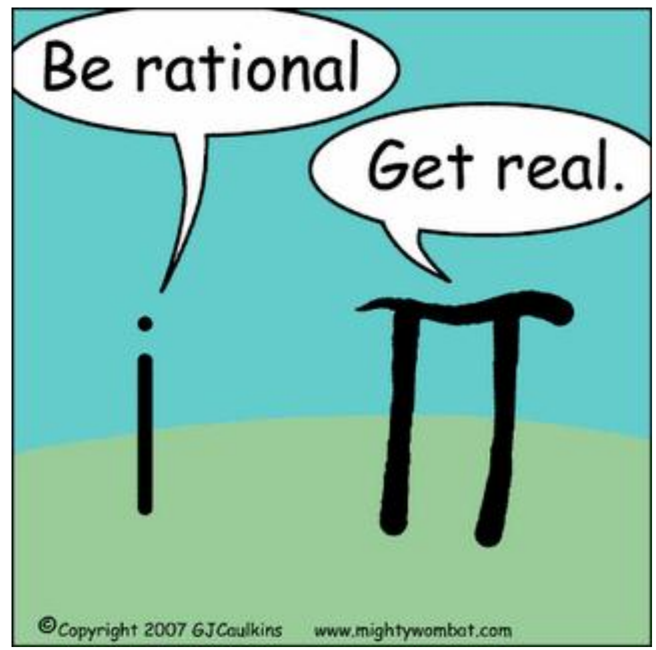
With material from Michael Ernst and Hal Perkins

Recall: Class Specifications

- Describe **abstract value:** what the class represents at an abstract level
  - What the client sees
  - What data the ADT holds
- Brief summary of the ADT
- Specfields: data fields of the ADT
  - e.g. length of Square
- Derived fields: data fields that can be computed from the spec fields
  - e.g. area = length^2 of Square

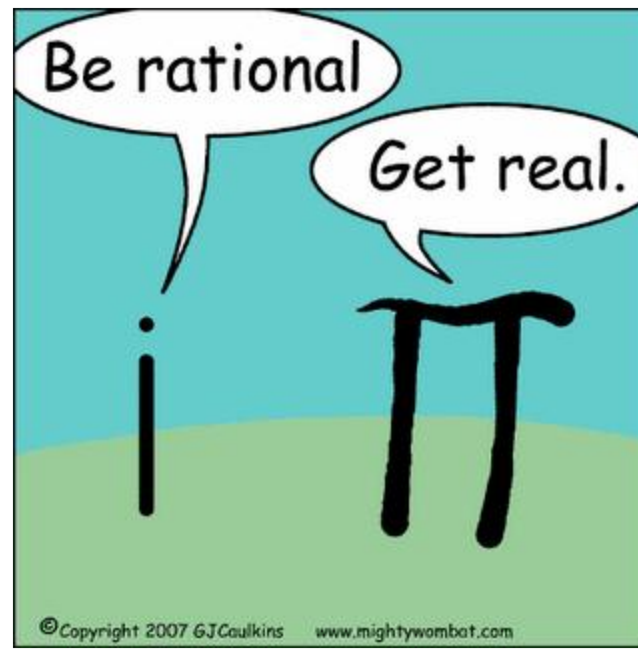
Example 1: Complex Number

- Specify a `ComplexNumber` class
- Represents number `a+bi`
- What are the abstract fields?
  - (What data does this class contain from the client's perspective?)



Complex Number

- Specify a `ComplexNumber` class
- Represents number `a+bi`
- What are the abstract fields?
  - **Real part, *a***
  - **Imaginary part, *b***



Let's formalize it

```
/**
 * ??????????????
 */
public class ComplexNumber {
     ...
}
```

Let's formalize it


```
/**
 * ?????????????
 */
public class ComplexNumber {
     ...
}
```

We don't need to know internal rep. to write client specs (why?)

Let’s formalize it


```
/**
 * ??????????????
 */
public class ComplexNumber {
      ...
}
```

We don’t need to know internal rep. to write client specs (why?)

See ComplexNumber1.java

Recall: Abstraction Function

- Specfields may not map directly to representation fields
  - Square has **length** specfield but not necessarily `private int length;`
- Internal representation can be anything as long as it somehow encodes the abstract value / specfields
- Abstraction function: a **mapping** from **internal state** to **abstract value**

Recall: Representation Invariant

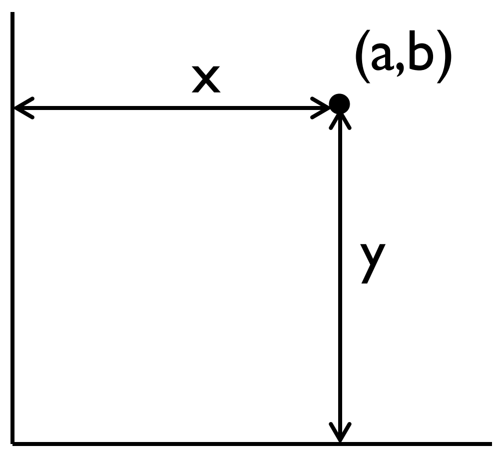
- Constrains an object's internal state
- Defines what must be true for abstraction function to hold
- If representation invariant is violated:
  - Object is "broken" – doesn't map to any abstract value

Let's implement ComplexNumber

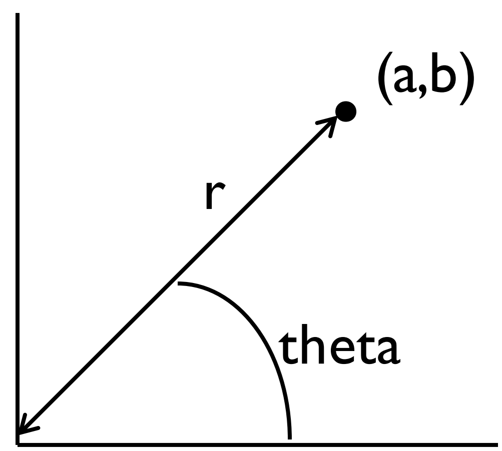
- Complex number often represented as point in Cartesian coordinate plane
- Possible representations:

(x, y)

(theta, r)



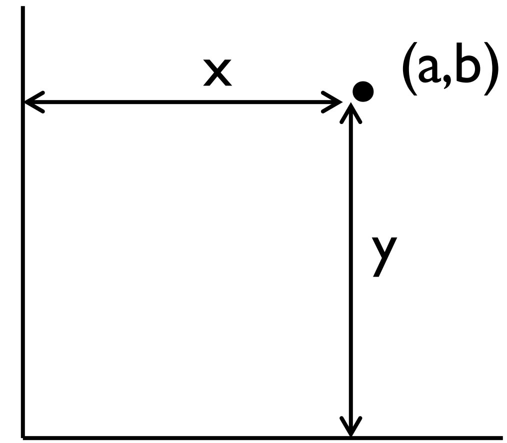
Cartesian coordinates



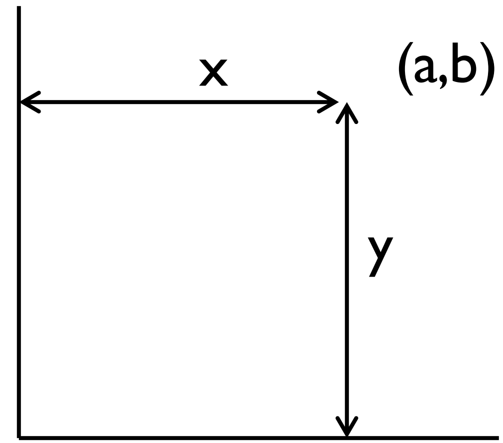
Polar coordinates

Implementation #1: Cartesian

- (x,y) coordinates
  - `x + y`i
- What is the AF?
- What is the RI?



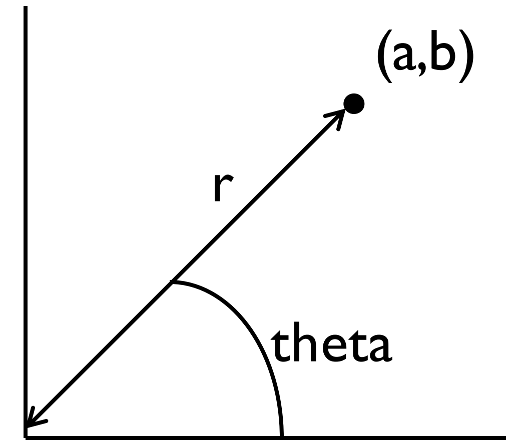
Implementation #1: Cartesian



- (x,y) coordinates
  - `x + yi`
- What is the AF?
- What is the RI?
  - RI is `true` – object cannot be in an invalid state!
- See [ComplexNumber1.java]

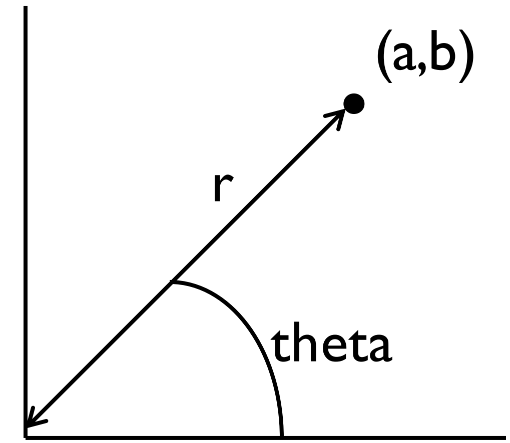
Implementation #2: Polar

- (theta, r)
  - `a: rad * cos(theta)`
  - `b: rad * sin(theta)`
- What is the AF?
- What is the RI?
- What should go in checkRep()?



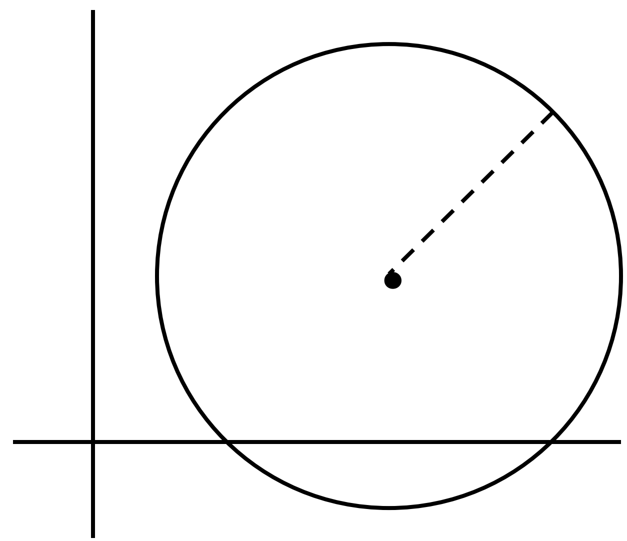
Implementation #2: Polar

- (theta, r)
  - `a: rad * cos(theta)`
  - `b: rad * sin(theta)`
- What is the AF?
- What is the RI?
- What should go in checkRep()?
- See ComplexNumber2.java



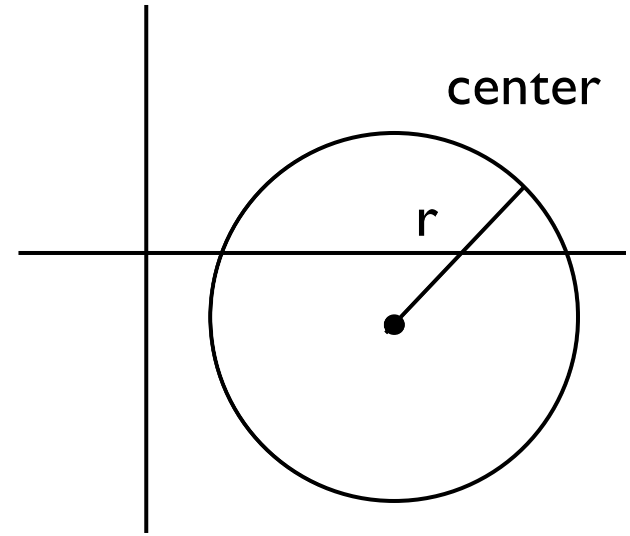
Example 2: Circle

- Circle on the Cartesian coordinate plane



Circle: Class Specification

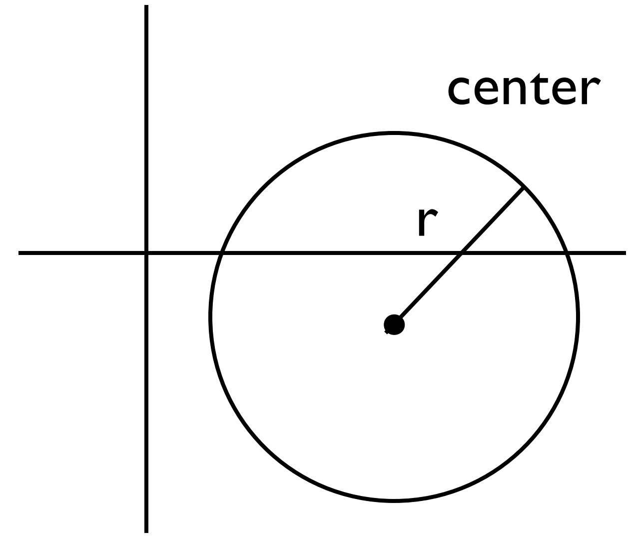
What are the abstract fields?



Circle: Class Specification

What are the abstract fields?

- Center point
- Radius
- Properties derived from these fields: circumference, area



Let’s formalize it

```
/**
 * ??????????????
 */
public class Circle {
     ...
}
```

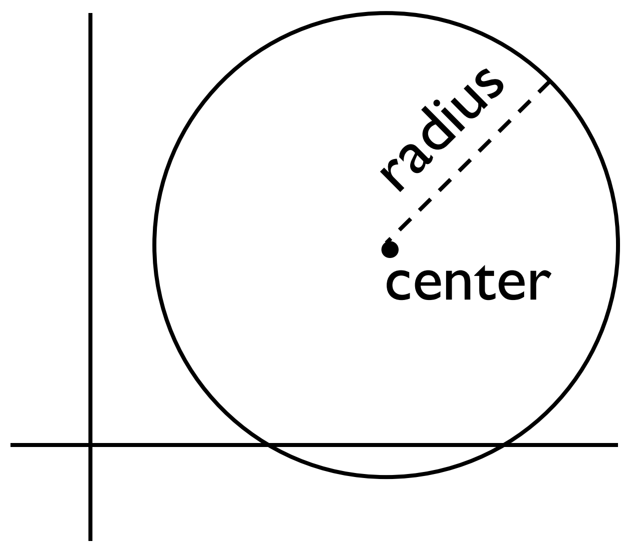
Let’s formalize it

```
/**
 * ??????????????
 */
public class Circle {
     ...
}
```

- See Circle1.java

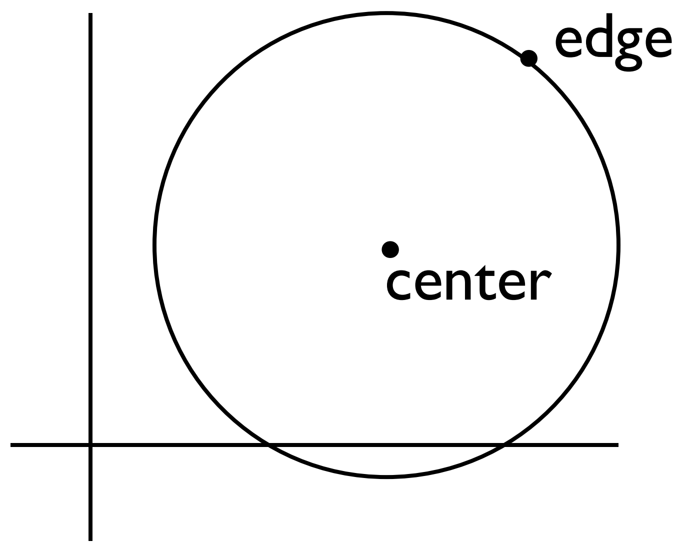
Writing AF, RI: Implementation 1

- Store center, radius directly
- Write the abstraction function, rep. invariant
- [Circle1.java]



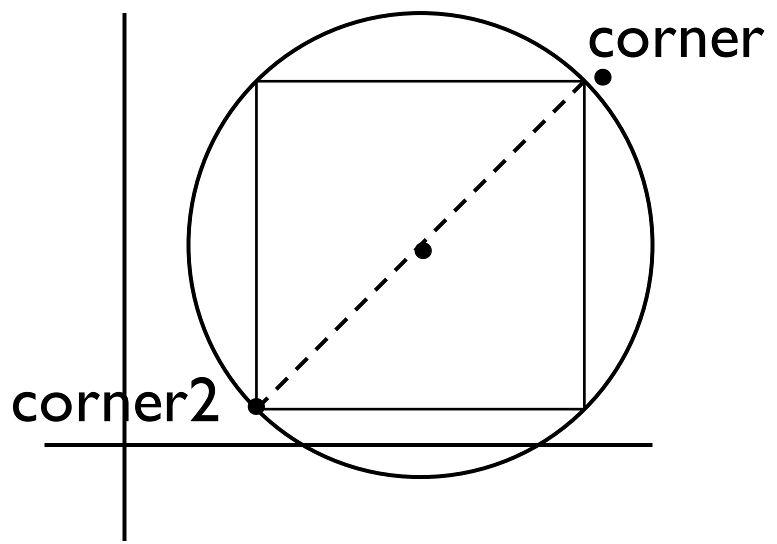
Writing AF, RI: Implementation 2

- Store center, edge point
- Write the abstraction function, rep. invariant
- Circle2.java



Writing AF, RI: Implementation 3

- Store corners of square inscribed in circle
- Write the abstraction function, rep. invariant
- Circle3.java



Example 3: Map

- Collection of <key, value> pairs
- Perform lookups by key
- What does the client see?

Example 3: Map

- Collection of <key, value> pairs
- Perform lookups by key
- What does the client see?
  - A collection of elements with some special properties – doesn't really have "specfields"
  - See IntTreeMap.java

Implementation: IntTreeMap

- DISCLAIMER: when *using* a map, TreeMaps are almost never what you want!
- HashMaps have much better performance
- But TreeMaps make a better AF/RI example here

Method Specifications

Precondition	@requires	determines the conditions under which the method may be invoked
Frame Condition	@modifies	a **list** of specfields identifying what might be modified by the method
Postcondition	@return	describes the value that gets returned, if any
	@throws	each of these lists an exception and the conditions under which it will be thrown
	@effects	any side effects that may result from invoking the method

General Guidelines

- Javadoc specs (/** … */) are **external** documentation
  - Visible to the client
  - Can be used to generate code-free documentation pages (e.g. Java API)
- So, Javadoc should only refer to what the client sees
  - Specfields / abstract value
  - Never instance fields or other internal details
- When referring to implementation details, use regular comments (//)
  - This includes **AF** and **RI**

General Guidelines, cont.

- Specs exist to help humans understand your code
- Crucial that they are easy to read and understand
- Be precise but concise
- Use formal mathematical notation or plain English – whichever is easier to understand