Object-Oriented Programming

CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

What Makes a Good Language?

Avoid common bugs (e.g., reading a nil pointer)

Ease of understanding

Ease of reuse

What Abstraction Means

Abstraction: omitting or hiding low-level details

Modularity: dividing up a system into components

Encapsulation: building walls around a module

Information hiding: hiding details of a module's implementation from the rest of the system

Separation of concerns: making a feature the responsibility of a single module

Abstract Data Types

Primitive types: values and operation on values

User-defined types: records, lists, …

Focus on values

ADT: defined by a set of operations on a type

Focus on operation

Stack is a type with `new, pop, push, empty` ...

Internal representation is less relevant

Classifying Operations

Creators: create new objects of type

Producers: create new objects from old ones

Mutators: change objects, e.g., `list.add(n)`

Observers: take objects of ADT and return objects with different type, e.g., `list.size()`

ADT Example

int

Creators: numeric literals `1`, `2`, `3`, …

Producers: arithmetic operations `+`, `-`, `*`, `/`, …

Observers: comparison operators `==`, `!=`, `<`, `>`

Mutators: none (immutable)

ADT Example

List

Creators: `ArrayList, LinkedList, …`

Producers: `Collections.unmodifiableList()`

Observers: `size(),get()`

Mutators: `add(),remove(), …`

ADT Example

String

Creators: `String(), String(char[])`

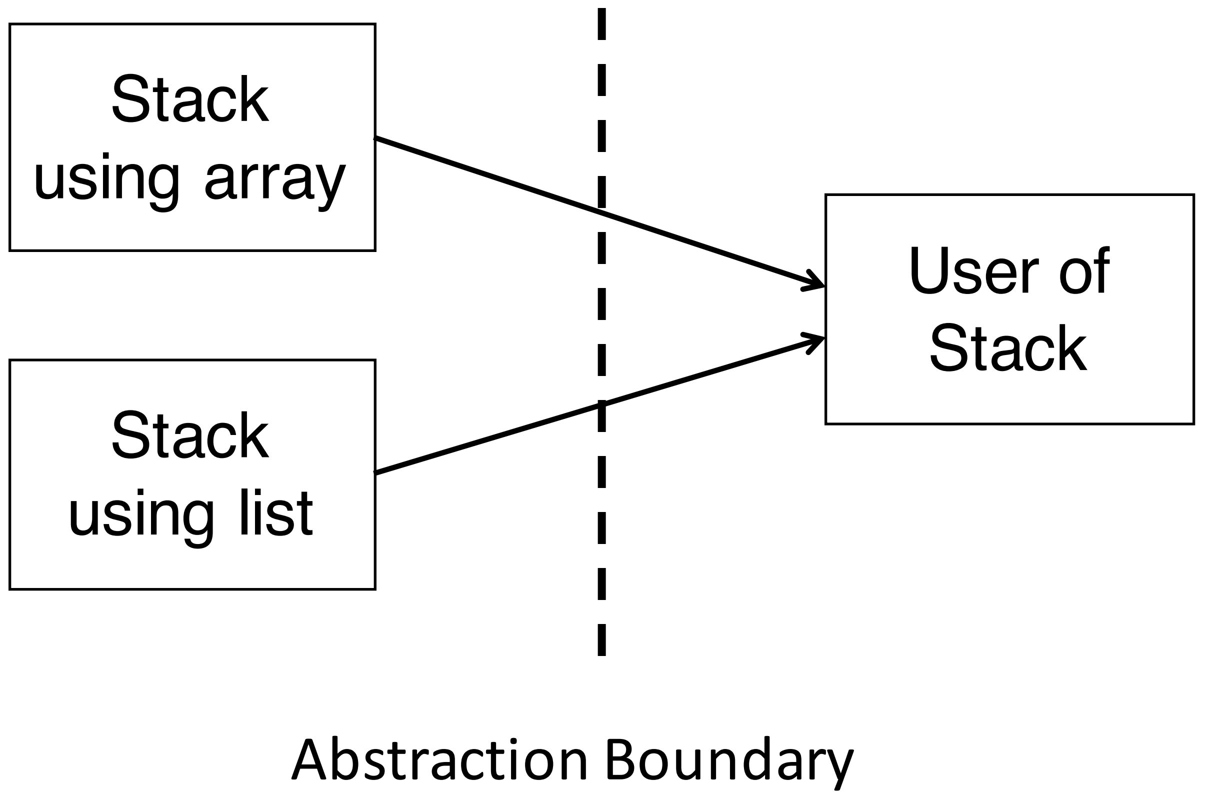
Producers: `concat(),substring(),`...

Observers: `length(), charAt(),`...

Mutators: none (immutable)

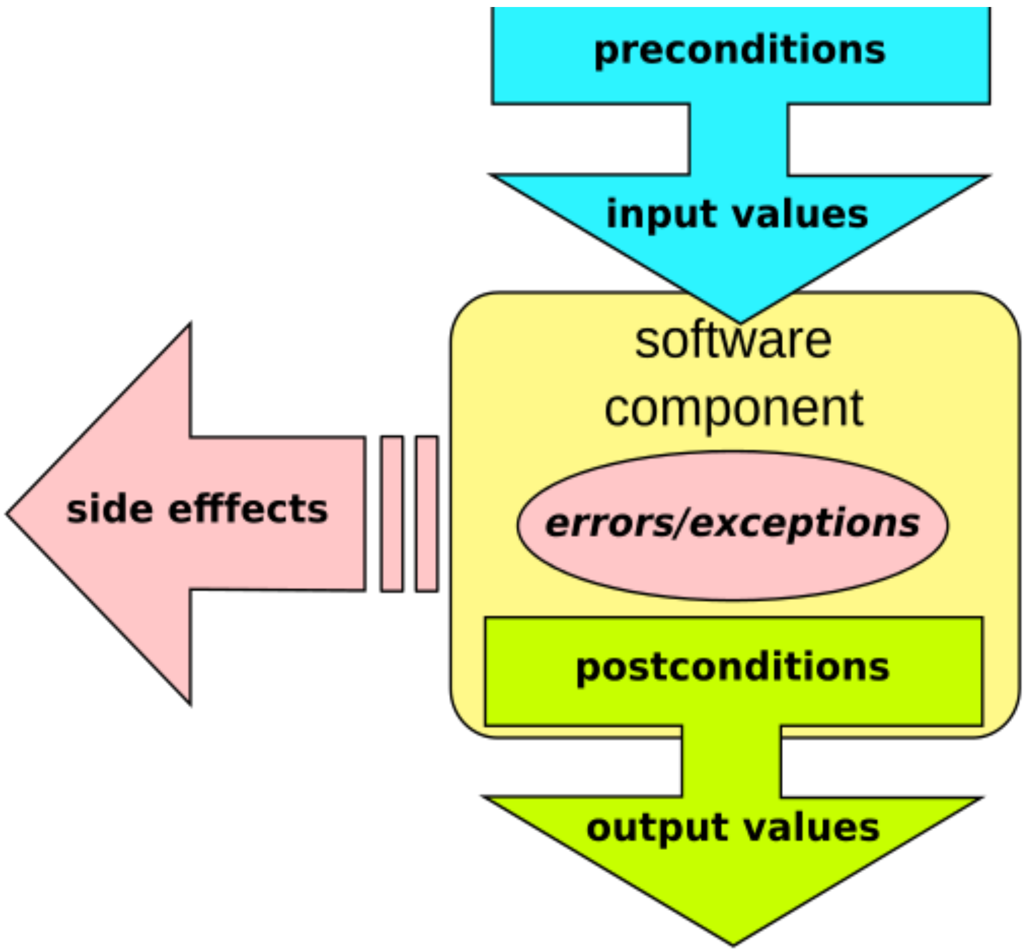
Representation Independence

A good ADT should be independent of impl.



Design by Contract

A good ADT should define a contract:



Contract

Precondition: assumptions on inputs

Side effects: changes to the value

Postcondition: functionality of the operation

Invariants: a property that is always true throughout the operation

Example: Rational Numbers

```
public class Rational {
    private int p,q; // represents p/q
    // class invariant: q > 0, gcd(p,q) = 1
    //                       Note: gcd(0,x) = x
```

Example: Rational Numbers

```
/** Create num/den.
    Requires: den != 0.
*/
public Rational (int num, int den) {
    if (den < 0) {
        num = -num;
        den = -den;
    }
    int g = gcd(num,den);
    p = num/g;
    q = den/g;
}
```

Precondition

Check den!=0?

Example: Rational Numbers

Side effects

```
/** Modifies: this to be this+r. */
public void add(Rational r) {
    int g = gcd(q, r.q);
    p = r.q/g * p + q/g * r.p;
    q *= r.q/g;
}
```

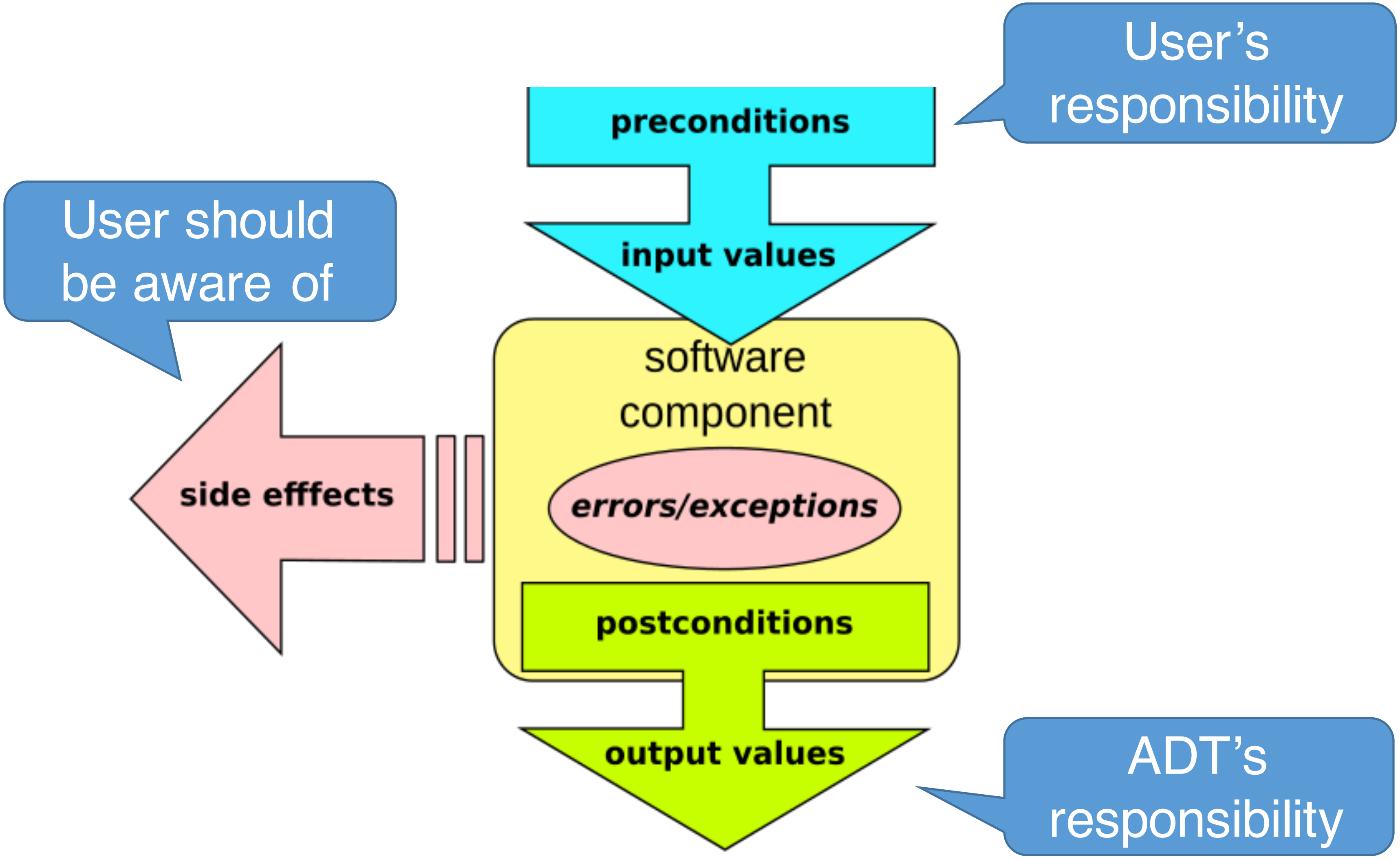
Check r.q!=0?

Example: Rational Numbers

Postcondition

```
 /** Returns x+y. */
public Rational plus(Rational x, Rational y) {
    Rational z = new Rational(x.p, x.q);
    z.add(y);
    return z;
}
```

Separation of Concerns



Enforcing Contract

How can we gain confidence that these contracts are all being obeyed?

Assertions: run-time condition check

```
assert (x > 0);
```

Stops the program if the condition is false

Programming with Assertions

```
public class Rational {
    private int p,q; // represents p/q
    // class invariant: q > 0, gcd(p,q) = 1
    //                  Note: gcd(0,x) = x
    boolean classInv() {
        return q > 0 && gcd(p, q) == 1;
    }
```

Programming with Assertions

```
/** Create num/den.
    Requires: den != 0.
*/
public Rational (int num, int den) {
    assert (den != 0);
    if (den < 0) {
        num = -num;
        den = -den;
    }
    int g = gcd(num,den);
    p = num/g;
    q = den/g;
    assert ClassInv();
}
```

Programming with Assertions

```
/** Modifies: this to be this+r. */
public void add(Rational r) {
    int g = gcd(q, r.q);
    assert (g != 0);
    p = r.q/g * p + q/g * r.p;
    q *= r.q/g;
    assert ClassInv();
}
```

Programming with Assertions

```
 /** Returns x+y. */
public Rational plus(Rational x, Rational y) {
    Rational z = new Rational(x.p, x.q);
    z.add(y);
    assert ClassInv();
    return z;
}
```

Assertion

Assertions are powerful weapons in catching bugs!

But they have performance overhead

Java: assertions are turn off by default

Pass in `-ea` to JVM to enable all assertions