CS 420 Advanced Programming Languages
Fall Semester, 2022
Doc 5 Dollar Words, Rust Modules
Sep 8, 2022

Review

Object-Oriented Programming is good as it promotes

- Code reuse
- More readable code
- More maintainable code
- Better designs

Abstraction

“Extracting the essential details about an item or group of items, while ignoring the unessential details.”

Edward Berard

“The process of identifying common patterns that have systematic variations; an abstraction represents the common pattern and provides a means for specifying which variation to use.”

Richard Gabriel

Encapsulation

Enclosing all parts of an abstraction within a container

Information Hiding

Hiding of design decisions in a computer program

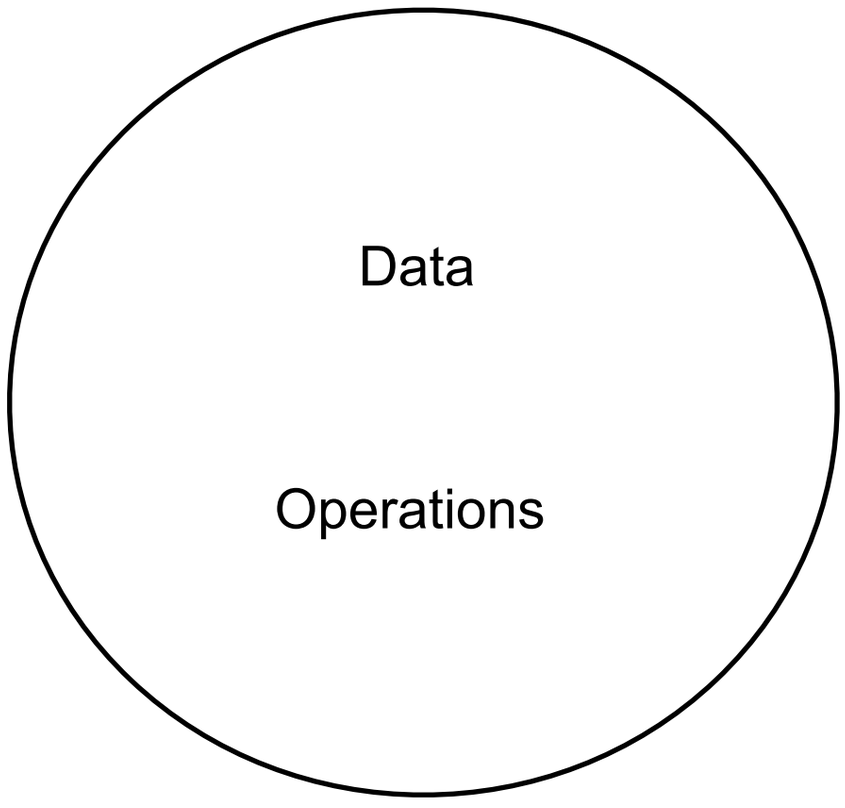
Hide decisions are most likely to change,
To protect other parts of the program

Class

Represents an abstraction

Encapsulates data and operations of the abstraction

Hide design decisions/details



Basic OO Heuristics

Keep related data and behavior in one place

A class should capture one and only one key abstraction

Beware of classes that have many accessor methods defined in their public interface

Non-OO items

Utility methods

Data classes

Utility method

Method in class that

- Does not access any field (data member, instance variables)
- Just uses parameters

Dollar Words

Character value

a, A -> 1
b, B -> 2
...
z, Z -> 26

Word value

Sum of the character values in the word

cab -> 3 + 1 + 2 = 6

Dollar Word

Word value = 100

buzzy	draftsmen
nutty	driveling
arrowy	dualities
crusts	ducklings
cutout	dumbfound
cutups	ebullient
drossy	ecstasies
dryrot	ejections
envoys	electives
flurry	elephants
grouts	elsewhere
growly	encumbers
grumpy	energiser
dominates	Englewood
	enticings
	equalized
	equipages

Problem

Given a string containing words find all the dollar words in the string

"This boulevard is in a status of tailspin!" "boulevard", "status", "tailspin"

Approach One

What do we have to do

- Separate the words
- Compute the value of each word
- Select the words with value 100

Using Java 8

```
public class DollarWorld {
    Set<Character> wordSeparators =
            Collections.unmodifiableSet(Stream.of(',', '.', '?', ':', ';','!', ' ').collect(toSet()));

    public ArrayList<String> separateWords(String text) throws IOException {
        ArrayList<String> words = new ArrayList<String>();
        StringBuffer currentWord = new StringBuffer();
        StringReader textReader = new StringReader(text);
        int next;
        while (( next = textReader.read()) != -1) {
            char nextChar = (char) next;
            if (wordSeparators(nextChar)) {                    // Collect chars until find word separator
                words.add(currentWord.toString());
                currentWord = new StringBuffer();
            } else {
                currentWord.append(nextChar);
            }
        }
        if (currentWord.length() >0 ) {                        // when at end of text likely to have
            words.add(currentWord.toString());                 // word collected in current Word
        }
        return words;
    }
}
```

```
public ArrayList<String> dollarWordsIn(ArrayList<String> words) throws IOException {
    ArrayList<String> dollarWords = new ArrayList<String>();

    for (String word:words) {
        int currentWordValue = 0;
        StringReader wordChars = new StringReader(word.toLowerCase());
        int next;

       // Compute word value
        while ((next = wordChars.read() ) != -1) {
           char nextChar = (char)next;
            int charValue = nextChar - 'a' + 1;
            if ((charValue >0) && (charValue < 27))
                currentWordValue += charValue;
       }
       if (currentWordValue == 100)
            dollarWords.add(word);
    }
    return dollarWords;
}
```

The Main Function

```
public ArrayList<String> dollarWordsIn(String text) throws IOException {
    return dollarWordsIn(separateWords(text));
}
```

```
@Test
void testDollarWords() throws IOException {
    String text = "This crusts is in a status of  truism!";
    List<String> dollarWords = new ArrayList<String>(Arrays.asList("crusts", "status", "truism"));
    DollarWorld testee = new DollarWorld();
    assertEquals(dollarWords, testee.dollarWordsIn(text));
}
```

Some More Tests

```
@Test
void separateWordsTest() throws IOException{
    ArrayList<String> correctAnswer = new ArrayList<String>(Arrays.asList("a", "b"));
    DollarWorld testee = new DollarWorld();
    assertEquals(correctAnswer, testee.separateWords("a b"));
}

@Test
void dollarWordsInTest() throws IOException {
    ArrayList<String> correctAnswer = new ArrayList<String>(Arrays.asList("status", "TRuISm"));
    ArrayList<String> input = new ArrayList<String>(Arrays.asList("foo", "status", "bar", "TRuISm"));
    DollarWorld testee = new DollarWorld();
    assertEquals(correctAnswer, testee.dollarWordsIn(input));
}
```

A Program in a Class

I wrote a program

Embedded it in a class

Issues

```
public class DollarWorld {
    Set<Character> wordSeparators =
            Collections.unmodifiableSet(Stream.of(',', '.', '?', ':', ';','!', ' ').collect(toSet()));
```

wordSeparators

- Not part of the state of the object
- Constant used in one method
- Make static or put in the method

```
public class DollarWorld {
    static Set<Character> wordSeparators =
            Collections.unmodifiableSet(Stream.of(',', '.', '?', ':', ';','!', ' ').collect(toSet()));
```

Block Comments often Indicate a Method

```
// Compute word value
while ((next = wordChars.read() ) != -1) {
   char nextChar = (char)next;
    int charValue = nextChar - 'a' + 1;
    if ((charValue >0) && (charValue < 27))
        currentWordValue += charValue;
}
```

Here is the Method

```
public int wordValue(String word) throws IOException {
    int wordValue = 0;
    StringReader wordChars = new StringReader(word.toLowerCase());
    int next;
    while ((next = wordChars.read() ) != -1) {
        char nextChar = (char)next;
        int charValue = nextChar - 'a' + 1;
        if ((charValue >0) && (charValue < 27))
            wordValue += charValue;
    }
    return wordValue;
}
```

Here is the Resulting dollarWordsIn method

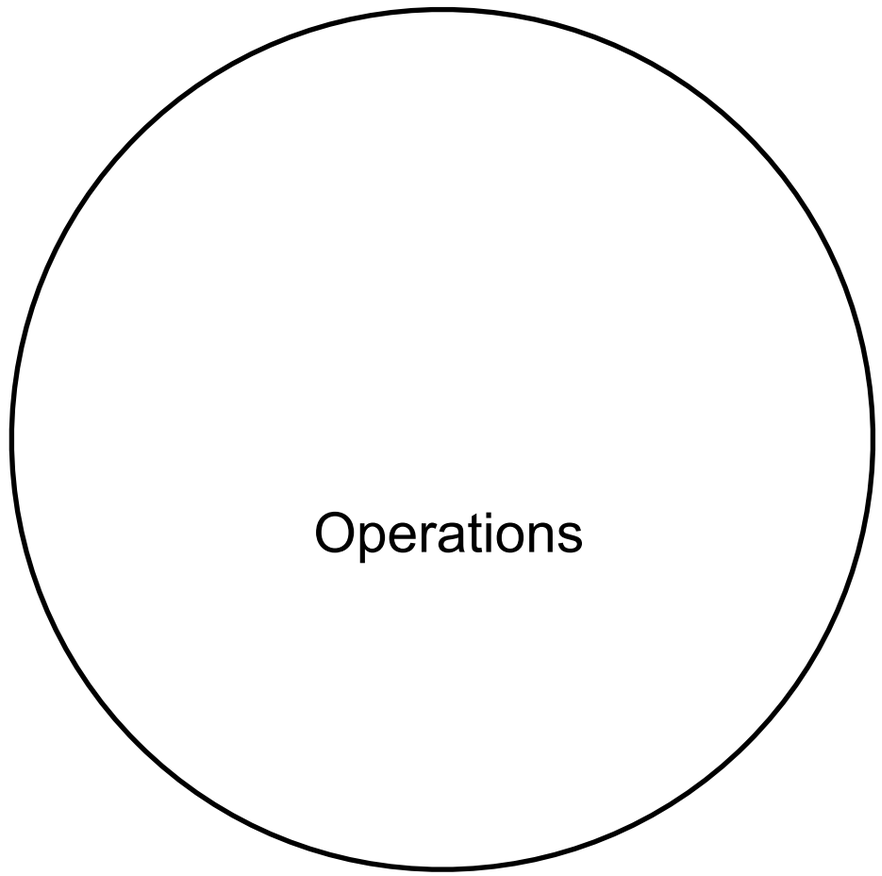
```
public ArrayList<String> dollarWordsIn(ArrayList<String> words) throws IOException {
    ArrayList<String> dollarWords = new ArrayList<String>();

    for (String word:words) {
       if (wordValue(word) == 100)
           dollarWords.add(word);
    }
    return dollarWords;
}
```

Issue: Code Reuse

Any Code reuse possible?

No Data

Just functions



A Different Approach

What building blocks can I make

What things need get done

Who should do them

Then write program

Kotlin

Regular function

```
fun decrement(value: Int) : Int {
    return value - 1;
}
```

```
@Test
fun decrementTest() {
    assertTrue( 2 == decrement(3));
}
```

Extension Methods

Can add methods to existing classes

Class the method is in

```
fun Int.inc(): Int {
    return this + 1;
}
```

```
@Test
fun incTest() {
    assertTrue( 2 == 1.inc());
}
```

Task: Convert Char to value

Char is the data, so add method to Char class

```
fun Char.wordValue(): Int {
    val wordValue = this.toLowerCase() - 'a' + 1
    return when {
        wordValue < 0 -> 0
        wordValue > 26 -> 0
        else -> wordValue
    }
}
```

```
 @Test
 fun testCharWordValue() {
     assertEquals(1, 'a'.wordValue())
     assertEquals(1, 'A'.wordValue())
     assertEquals(26, 'Z'.wordValue())
     assertEquals(0, '+'.wordValue())
 }
```

Task: Compute value of a String

String (CharSequence) is the data, so add method to CharSequence interface

```
fun CharSequence.wordValue(): Int {
    return this.sumBy { it.wordValue() }
}
```

```
@Test
fun testCharSequenceWordValue() {
    assertEquals(2, "aa".wordValue())
    assertEquals(6, "abc".wordValue())
    assertEquals(0, "".wordValue())
    assertEquals(100,"buzzy".wordValue())
}
```

Task: Determine if String is Dollar Word

String (CharSequence) is the data, so add method to CharSequence interface

```
fun CharSequence.isDollarWord(): Boolean {
    return this.wordValue() == 100
}

@Test
fun testIsDollarWord() {
    val dollarWords = listOf("crusts", "status", "truism","TRuISm", "comport", "grouper")
    dollarWords.forEach {
        assertTrue { it.isDollarWord() }
    }

    val nonDollarWords = listOf("cat", "", "Mouse")
    nonDollarWords.forEach {
        assertFalse { it.isDollarWord() }
    }
}
```

Task: Determine if Char separates Words

Char is the data, so add method to Char class

```
fun Char.isWordSeparator(): Boolean {
    val separators = setOf(',', '.', '?', ':', ';','!', ' ')
    return separators.contains(this)
}
```

```
@Test
fun testIsSeparator() {
    assertTrue(','.isWordSeparator())
    assertFalse('q'.isWordSeparator())
    assertTrue(' '.isWordSeparator())
}
```

Task: Break String into Parts

String is the data, so add method to CharSequence interface

```
fun CharSequence.separatedBy(separator: (Char) -> Boolean): List<CharSequence> {
    val words = mutableListOf<CharSequence>()
    val currentWord = StringBuilder()

    this.forEach {
        if (separator(it)) {
            words.add(currentWord.toString())
            currentWord.clear()
        } else {
            currentWord.append(it)
        }
    }
    if (currentWord.isNotEmpty())
        words.add(currentWord.toString())
    return words
}
```

Task: Find all Dollar Words in a String

String is the data, so add method to CharSequence interface

```
fun CharSequence.dollarWords(): List<CharSequence> {
    val words = this.separatedBy { it.isWordSeparator()  }
    return words.filter { it.isDollarWord() }
}
```

```
@Test
fun testDollarWords() {
    val text = "This crusts is in a status of  truism!"
    val dollarWords = listOf("crusts", "status", "truism")
    assertEquals(dollarWords, text.dollarWords())
}
```

Code Reuse Possible?

Most method are specific to Dollar words

Any can easily be used elsewhere

isWordSeparator

Might be useful elsewhere

separatedBy

More likely to be used elsewhere

Part of OO Design Process

Who is on the team?

What are the goals of the system?

What must the system accomplish?

What objects are required to model the system and accomplish the goals?

What are their tasks, responsibilities?

What does each object have to know in order to accomplish each goal it is involved with?

What steps toward accomplishing each goal is it responsible for?

Modules

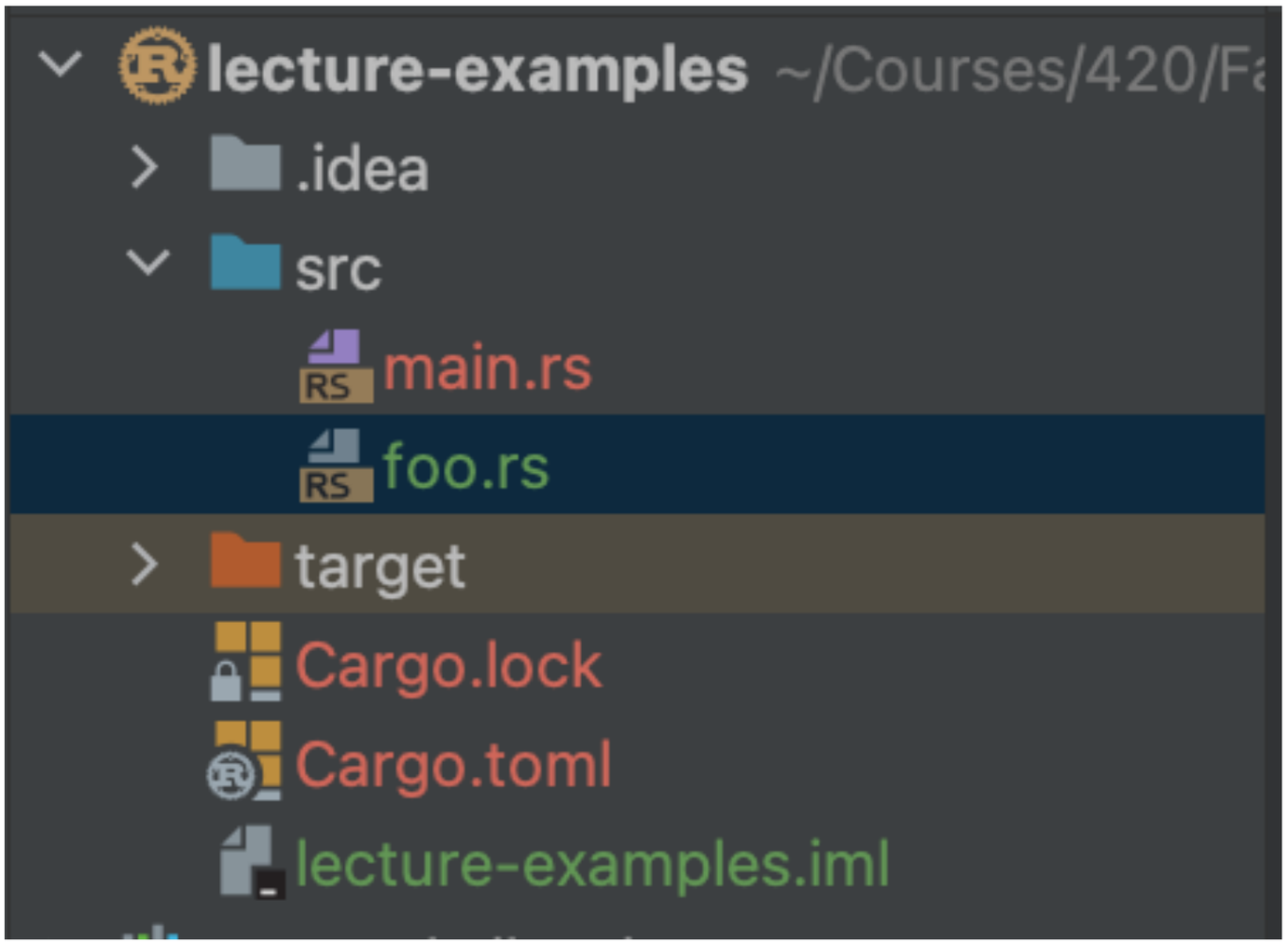
Like Java package - namespace

Three ways to define or organize modules

foo.rs

```
pub struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}

pub trait Summary {
    fn summarize(&self) -> String {
        String::from("(Read more...)")
    }
}
```

```
impl Summary for Rectangle {
    fn summarize(&self) -> String {
        format!("{}x{}", self.width, self.height)
    }
}

impl Summary for i32 {
    fn summarize(&self) -> String {
        format!("i32: {}", self)
    }
}
```


lecture-examples ~/Courses/420/F
.idea
src
main.rs
foo.rs
target
Cargo.lock
Cargo.toml
lecture-examples.iml

main.rs

```
mod foo;

fn main() {
    let rect = foo::Rectangle { width: 30, height: 50 };
}
```

Compile Error - width & height are private fields

Solution 1

foo.rs

```
pub struct Rectangle {
    pub width: u32,
    pub height: u32,
}

//etc
```

main.rs

```
mod foo;

fn main() {
    let rect = foo::Rectangle { width: 30, height: 50 };
}
```

```
mod foo;
use foo::Rectangle;

fn main() {
    let rect = Rectangle { width: 30, height: 50 };
}
```

Solution 2 - Constructor

foo.rs

```
pub struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    pub fn new(width: u32, height: u32) -> Rectangle {
        Rectangle { width, height }
    }

    fn area(&self) -> u32 {
        self.width * self.height
    }
}
```

main.rs

```
mod foo;

fn main() {
    let rect = foo::Rectangle::new(30, 50 );
}
```

Using the Trait

```
mod foo;
use crate::foo::Summary;

fn main() {
    let rect = foo::Rectangle::new(30, 50 );
    println!("rect: {}", rect.summarize());
    println!("i32: {}", 42.summarize());
}
```

foo.rs

```
pub struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    pub fn new(width: u32, height: u32) -> Rectangle {
        Rectangle { width, height }
    }

    pub fn area(&self) -> u32 {
        self.width * self.height
    }
}

pub trait Summary {
    fn summarize(&self) -> String {
        String::from("(Read more...)")
    }
}
```

```
impl Summary for Rectangle {
    fn summarize(&self) -> String {
        format!("{}x{}", self.width, self.height)
    }
}

impl Summary for i32 {
    fn summarize(&self) -> String {
        format!("i32: {}", self)
    }
}
```

Passing Traits as Parameter

```
fmod foo;
mod tests;

use crate::foo::Summary;

fn bar(test: &impl Summary) {
    println!("{}", test.summarize());
}
```

```
fn main() {
    let rect = foo::Rectangle::new(30, 50 );
    bar(&rect);
    bar(&42);
}
```