Week 4 Discussion

CS 131 Section 1B
22 April 2022
Danning Yu

Announcements

- Midterm next Thursday 4/28
  - Sample midterms available on BruinLearn
- HW3 released, due 5/4
- Homeworks should be submitted on BruinLearn, under Assignments
- Before submitting
  - Make sure your code compiles on SEASnet server
  - Make sure your function signatures are correct
  - Follow all instructions and specifications
  - Do not submit files in a .zip unless told to do so
- Help and starter code from past TAs
  - https://github.com/CS131-TA-team

Midterm

Topics Covered

- Textbook: chapters 1-11 (ML for OCaml); 13, 15, 17 (Java)
- Homeworks 1-3
- Lectures: all
- C/C++ basis, type & inheritance etc. are also included
- There might be some open-ended questions with no fixed answer

Topics Covered - Languages

- OCaml
  - Reading and writing code, figuring out types
- Java
  - Concept of OOP features
  - Java types, generics, subtypes, etc.
  - Java Memory Model, race conditions, concurrency
- Grammar (Syntax)
  - Representations: BNF, EBNF, syntax diagrams, …
  - Derivation and ambiguity

Fall 2020 Question 1

1. Consider the quiz given on the first day of class, in which M.D. McIlroy wrote a shell script that computed a concordance of words found in the input, sorted by popularity of occurrence. You want an OCaml function ‘mcilroy’ that does what McIlroy’s script did, but using OCaml objects instead of POSIX character streams. ‘mcilroy’ should take a sequence of characters as an argument, and return a concordance that contains the same information as the concordance output by McIlroy’s shell command. You may assume the existence of functions ‘sort’, ‘tr’, and ‘uniq’ that have the functionality of their POSIX counterparts, but which are OCaml functions instead of being POSIX utilities.

- An open ended question
- The command to "convert" into OCaml
  - `tr -cs 'A-Za-z' '[\n*]' | sort | uniq -c | sort -rn`

Fall 2020 Question 1

1a (8 minutes). Give the types of the functions ‘mcilroy’, ‘sort’, ‘tr’, and ‘uniq’. Base the types on ‘char’, not ‘string’; for example, if you want a finite sequence of characters, use a list or a tuple of ‘char’, not ‘string’.

1b (8 minutes). Implement ‘mcilroy’ in terms of the other functions.

- What you want to do: specify interfaces and types clearly
  - Considerations: How to express command line arguments? I/O?
  - Multiple answers possible, accepted as long as it makes sense.
  - Abstraction: no need to worry how the “commands” are implemented

Fall 2020 Question 2

2 (6 minutes). Give an example of Java code that is not referentially transparent.

- Keyword: what does referentially transparent mean?

Fall 2020 Question 2

2 (6 minutes). Give an example of Java code that is not referentially transparent.

```
int i = 0
int incrementAndGet(){
    i++;
    return i;
}

incrementAndGet(); // returns 1
incrementAndGet(); // returns 2 - every time we call
                   // incrementAndGet the value
                   // returned changes
```

Fall 2020 Question 3 - OCaml Types

- Create a higher-order function `r3` that reverses the sequence of arguments of 3-arguments function
  - e.g. let rf = r3 f, then `rf c b a` should act like `f a b c`
  - Answer: `let r3 f a b c = f c b a`
- Can you create a generalized version (called `rn`) that works for any number of arguments in OCaml?
  - e.g. `r3` could be implemented as `let r3 = rn 3`, similar for `r2`, `r4`, …
  - Answer: no, because OCaml must determine function types at compile time, so there cannot be a variable number of arguments

Fall 2020 Question 4 - Extending HW2

4 (9 minutes). In Homework 2, the definition for “alternative list” said, “By convention, an empty alternative list [] is treated as if it were a singleton list [[]] containing the empty symbol string.” Suppose instead you had been assigned a variant Homework 2V in which the definition had said “By convention, an empty alternative list [] means that nothing can be derived from the nonterminal in question; i.e., the nonterminal is a blind alley that can never be used to produce a sentence in the grammar”, and suppose you had written a function “make_parserV” that acted like “make_parser” except it solved Homework 2V instead of Homework 2.

Would it be reasonable to implement make_parser in terms of make_parserV? or make_parserV in terms of make_parser? or both? or neither? Briefly justify your answer.

Fall 2020 Question 5a - Grammar Ambiguity

```
syntax = syntax rule, {syntax rule};
syntax rule = meta identifier, '=', definitions list, ';';
definitions list = single definition, {'|', single definition};
single definition = primary, {',', primary};
primary = optional sequence | repeated sequence | special sequence
  | grouped sequence | meta identifier | terminal string | empty;
empty = ;
optional sequence = '[', definitions list, ']';
repeated sequence = '{', definitions list, '}';
special sequence = '?', {character}, '?';
grouped sequence = '(', definitions list, ')';
terminal string = "'", character, {character}, "'"
  | '"', character, {character}, '"';
meta identifier = letter, {letter | decimal digit};
```

```
5a (5 minutes). Show that this grammar is ambiguous.  Assume the
usual way of expanding repetitions like {syntax rule} into plain BNF.
```

- A grammar is ambiguous if you can create 2 different parse trees for the same final result
- If a question is hard, skip it and revisit it later

Fall 2020 Question 5b - Syntax Diagram

5b (10 minutes). Translate this grammar into syntax diagrams in the style of Webber, circling nonterminals and drawing boxes around terminals. Keep your diagrams simple by eliminating any nonterminal that is used only once (except do not eliminate the start symbol ‘syntax’). You need not diagram ‘letter’, ‘character’, or ‘decimal digit’.

- Grammar definition style appeared in HW1 and 2
- Parse tree
- BNF, EBNF
- Syntax diagram
  - Syntax chart, syntax graph, “railroad diagram” etc.
- Abstract syntax tree (AST)

Fall 2020 Question 5 - BNF

- A BNF grammar contains:
  - The set of tokens (terminal symbols in HW1/2)
  - The set of non-terminal symbols (nonterminal symbols in HW1/2)
  - The start symbol (non-terminal)
  - The set of productions (rules in HW1/2)
    - `left-hand-size ::= right-hand-side`
  - LHS should be non-terminal symbol, RHS is a sequence of token or non-terminal
- `<expr> <if-stmt> <else-part>`
- Use `<empty>` for an empty RHS:
  - `<silent-example> ::= <empty>`

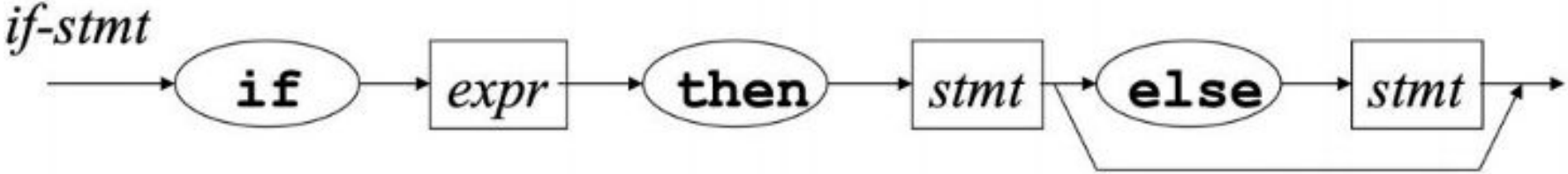
Fall 2020 Question 5 - EBNF

- Additional syntax to make grammar more terse
- `{x}`: zero or more repetitions of x (also written as `x*`)
- `[x]`: zero or one occurrence of x (also written as `x?`)
- `x+`: Kleene closure, i.e. one or more occurrences
- `( )`: grouping / precedence
- `=`: definition
- `(* ... *)`: comments
  - Same syntax as OCaml

Fall 2020 Question 5 - Syntax Diagram

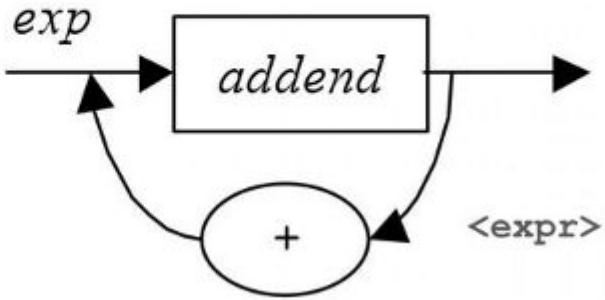
```
<if-stmt> ::= if <expr> then <stmt> else <stmt>
```


```
<if-stmt> ::= if <expr> then <stmt> [ else <stmt> ]
```

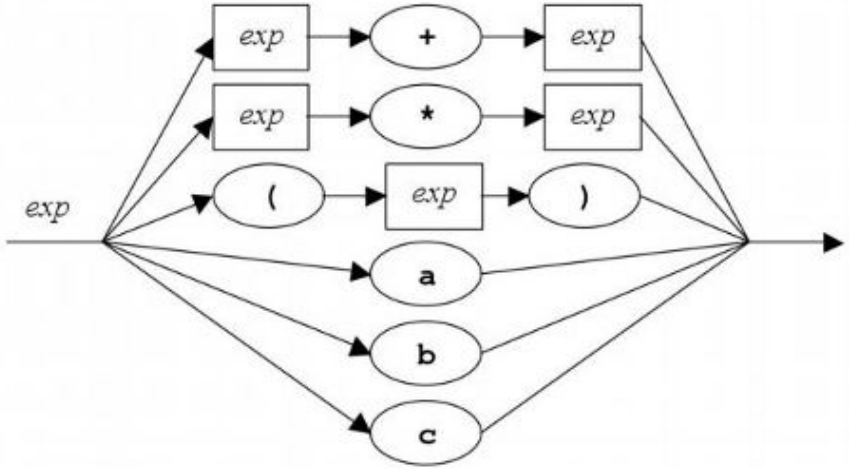


Fall 2020 Question 5 - Syntax Diagram

```
<expr> ::= <addend> {+ <addend> }
```



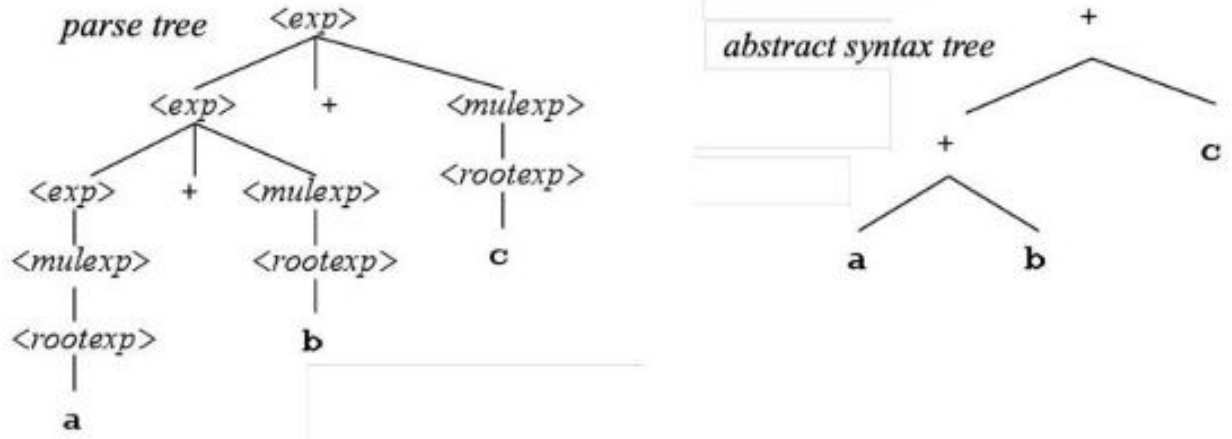

```
<expr> ::= <expr> + <expr> | <expr> * <expr> | ( <expr> )
         | a | b | c
```



Fall 2020 Question 5 - Parse Tree, AST

- AST: abbreviated version of parse tree
  - Details are implementation-dependent
  - Usually a node for every operation, with subtrees being corresponding operands

E.g. `(a + b) + c`



Fall 2020 Question 6a - Java Memory Model

```
int v = 10;
boolean flag = false;
void setem () { v = 20; flag = true; }
void getem () { if (flag) System.out.println("v = " + v); }
```

Suppose thread 1 executes ‘setem’ at about the time that thread 2 executes ‘getem’, and that these are the only uses of any code that accesses the variables ‘v’ and ‘flag’. Explain whether the output could be "v = 10" under the Java Memory Model.

- Make sure you understand the Java Memory Model and meanings of keywords like `synchronized` and `volatile`
- Answer: yes, it's possible, as code reordering might occur for `setem()` and the threads can execute in any order

Fall 2020 Question 6b & 6c - Java Memory Model

```
int v = 10;
boolean flag = false;
void setem () { v = 20; flag = true; }
void getem () { if (flag) System.out.println("v = " + v); }
```

6b (4 minutes). How would your answer to (a) differ if ‘setem’ and ‘getem’ were declared synchronized instead? Briefly explain.

6c (6 minutes). How would your answer to (a) differ if ‘v’ was declared volatile instead? (Assume ‘synchronized’ is not used.) Briefly explain.

- 6b answer: race condition is fixed
- 6c answer: race condition not fixed. The variable with race condition is `flag`; declaring `v` as volatile is not helpful towards fixing the race condition (need to declare `flag` as `volatile`)

Fall 2020 Question 7 - Write a Function

7 (12 minutes). Write a function adjdup that takes a list and returns a list of the same length with the same elements in the same order, except that if there are duplicates in the original list, they are reordered to be just after the first of the duplicates. For example, adjdup [7;6;7;8;8;4;10;4;3;5;1;2;7;7;10;9;8;5] should returns [7;7;7;7;6;8;8;8;4;4;10;10;3;5;5;1;2;9]. Your implementation may use the Stdlib and List modules, but it should use no other modules.

Fall 2020 Question 7 - Write a Function

```
let rec adjdup list = match list with
    | [] -> []
    | car::cdr -> (List.filter (fun a -> a = car) list)
            @ adjdup (List.filter (fun a -> a != car) cdr)
```

```
let rec adjdup l = match l with
    [] -> []
  | hd::tl ->
    let eq_l = List.filter ((=) hd) l
    and ne_l = List.filter ((<>) hd) l
    in eq_l @ (adjdup ne_l)
```

```
let rec adjdup l = match l with
    [] -> []
  | hd::tl ->
    let (eq_l, ne_l) = List.partition ((=) hd) l
    in eq_l @ (adjdup ne_l)
```

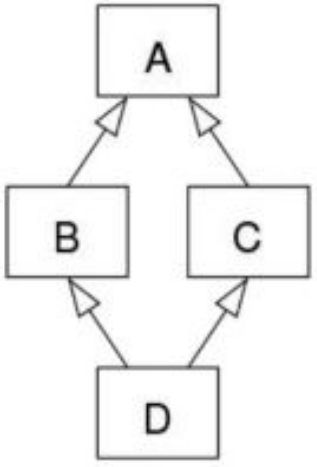
Write a Function - Advice

- Try to have a clear description of algorithm before writing code, understand what exactly is being asked
- Take advantage of OCaml language features and allowed library functions to make your code expressive
- Focus on clarity; efficiency is often secondary (but not unimportant)
- Make your code compiles!

Fall 2020 Question 8 - Java Classes

8 (6 minutes). In Java, a class can implement as many interfaces as you like, but it can extend at most one class. Suppose we invented a language Avaj which was just like Java except that in Avaj a class can extend as many classes as you like, but it can implement at most one interface. Explain why Avaj would be more problematic than Java in practice.

- Answer
  - Diamond inheritance, where classes B and C extend from A and class D extends both B and C, can cause issues
  - If B and C both implement a method that D does not, and then we call this method with an object from D, it is unclear if B or C's method should be called



Diamond Problem

Other Questions By Topic

- OCaml write a function
  - Fall 2017 1a, Spring 2008 2ab, 7ab
- OCaml typing
  - Fall 2017 1bc, Spring 2008 2c, 6, 7c
- Extensions of homework
  - Fall 2017 6 - requires you to understand the old HW2
- Java subtyping and inheritance
  - Fall 2017 3ab, Spring 2008 3d, 5
- Java memory model
  - Fall 2017 5
- Grammar
  - Basics: Fall 2017 4, Spring 2008 3d
  - Ambiguity: Spring 2008 3bc

Java and Homework 3

Pictures and diagrams borrowed from Boyan Ding and others

Object Oriented Programming

- Main idea: objects with methods and fields
  - Methods and fields are functions and variables that belong to the object, and encapsulated within
  - Object of the same class share same fields and methods
- Most popular programming paradigm
  - Java, C++, C#, Python, PHP, JavaScript, Ruby, Objective-C, Swift, Scala, Common Lisp, Smalltalk, …
  - i.e. Most of the popular languages today
- Possible benefits
  - Modularity
  - Information-hiding
  - Code reuse
  - Pluggability and ease of debugging
  - And more

Classes and Interfaces

- Class: template for an object
  - Object is an instance of a class
  - e.g. We can have multiple Bicycle objects that function the same way, but can be moving at different speed etc.
- All objects created using the same class will have the same methods/fields
- Interface: a description of what needs to be implemented
  - Multiple classes can implement the same interface
  - In Java, a list of functions
  - Allows for separation of API from implementation

Alan Kay's Definition of OOP

- Everything is an object
  - Numbers, classes, functions, ...
- Objects communicate by sending/receiving messages
  - Think of biological cells communicating
- Objects have their own memory
- Every object is an instance of some class
- All objects of the same type can receive the same messages

Some of these do not apply to most modern OOP languages!

Java

- General-purpose, object-oriented language
- One of the most popular programming languages
- Code compiled into bytecode and runs on a virtual machine
  - What are the pros and cons of this?
- Popular IDEs include Eclipse, Intellij IDEA
  - We don’t require usage of IDE, you can use any text editor for your homework

Java: Hello World

- `HelloWorld.java`

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, world");
  }
}
```

- How to compile
  - `javac HelloWorld.java`
  - Generates `HelloWorld.class` containing bytecode
- Running
  - `java HelloWorld`
  - Note: use class name, not file name

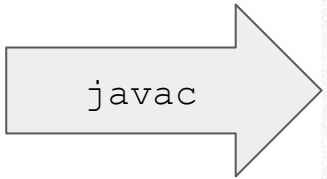
Files in Java

- `MyClass.java`: code for `MyClass`
- `MyClass.class`: bytecode for `MyClass` (compiled from `Myclass.java`)
- `Foo.jar`: Java Archive File
  - Is really just a ZIP archive
  - Often used to package whole compiled application with resources, configuration, etc
  - Will use this to package source files for Homework 3

Java Bytecode

- A intermediate form between compiled and interpreted code
  - Platform independence of interpreted code
  - Better performance than interpreted code

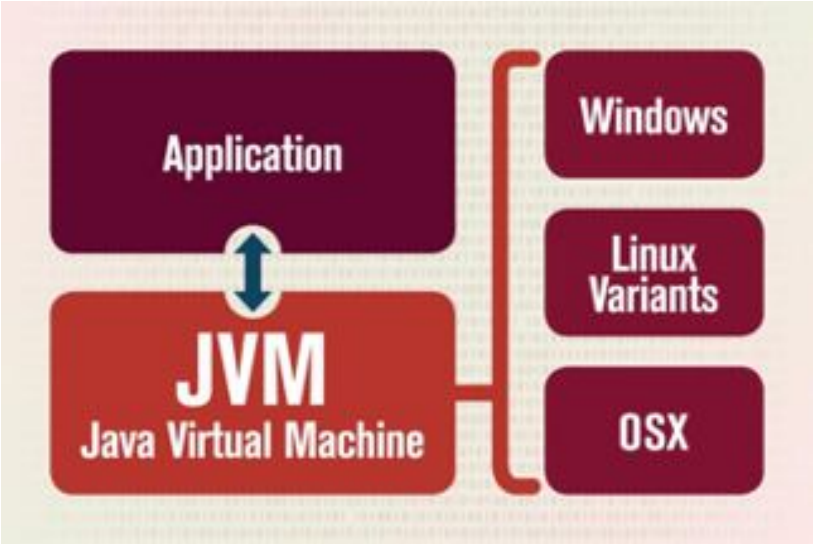
```
outer:
for (int i = 2; i < 1000; i++) {
    for (int j = 2; j < i; j++) {
        if (i % j == 0)
            continue outer;
    }
    System.out.println (i);
}
```




```
0:  iconst_2
1:  istore_1
2:  iload_1
3:  sipush  1000
6:  if_icmpge       44
9:  iconst_2
10: istore_2
11: iload_2
12: iload_1
13: if_icmpge       31
16: iload_1
17: iload_2
18: irem
19: ifne    25
22: goto    38
25: iinc    2, 1
28: goto    11
31: getstatic       #84;
34: iload_1
35: invokevirtual   #85;
38: iinc    1, 1
41: goto    2
44: return
```

Java Virtual Machine (JVM)

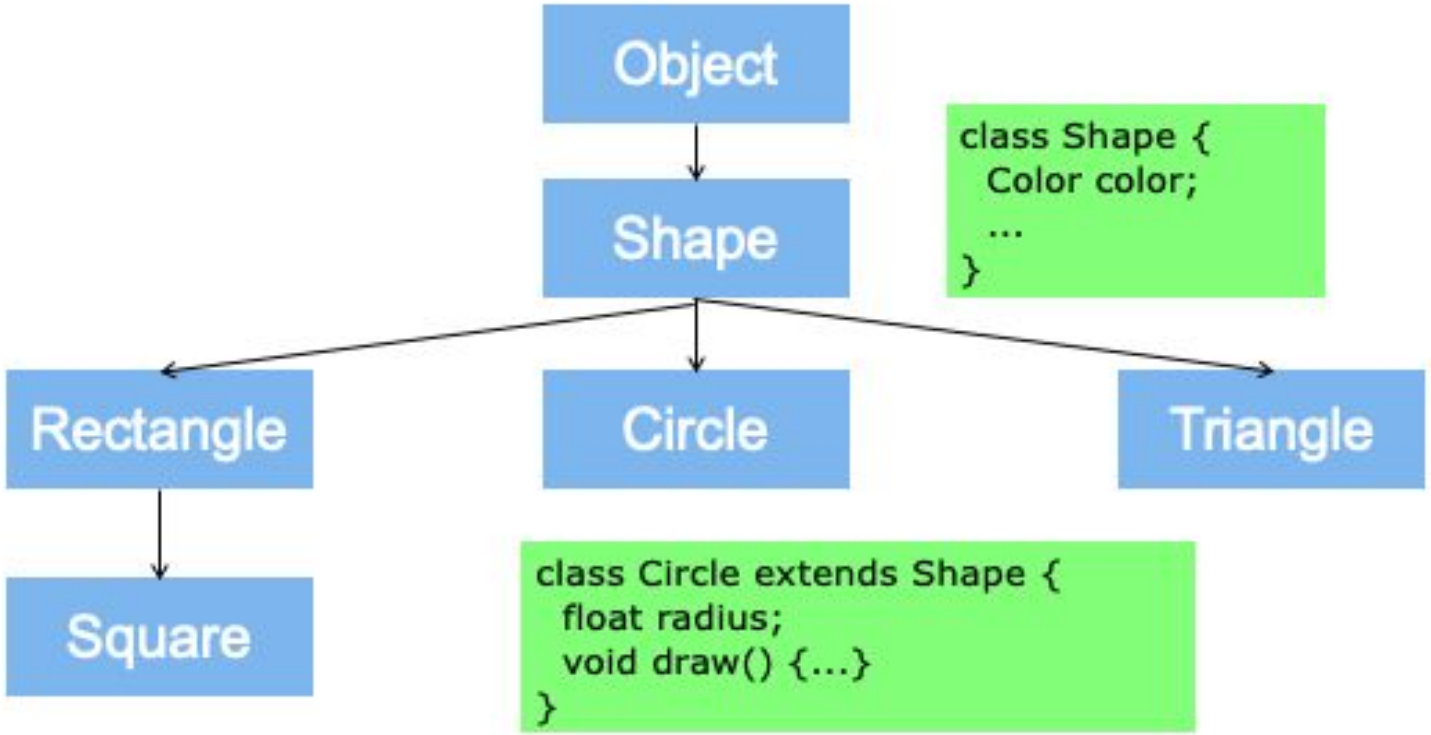
- Runs bytecode generated by a Java compiler
- Provides separation of code and operating system
  - Allows Java code to run on a variety of OSes
- JVM provides garbage collection, just-in-time compilation (JIT), etc
- Multiple implementations
  - Reference implementation (OpenJDK) provided by Oracle



OOP In Java

- Abstraction
- Encapsulation
  - Binding data with code that manipulates it
  - Access modifiers: `public/protected/private`
- Inheritance
  - An object may acquire some/all property of another object
- Polymorphism
  - One method can have multiple implementations, usage decided at runtime

Inheritance in Java



Inheritance in Java

```
class Shape {
    void draw() { /* do nothing */ }
}
class Rectangle extends Shape {
    void draw() { /* draw a rectangle */
}
}
class Circle extends Shape {
    void draw() { /* draw a circle */ }
}
class Triangle extends Shape {
    void draw() { /* draw a triangle */ }
}
```

```
Triangle a = new
Triangle();
/* draws a triangle */
a.draw();

Shape b = a;
/* draws a triangle */
b.draw();

b = new Circle();
/* draws a circle */
b.draw();
```

Inheritance in Java

- Which of the following are allowed?

```
Square a = new Square();
Shape b = a;
```

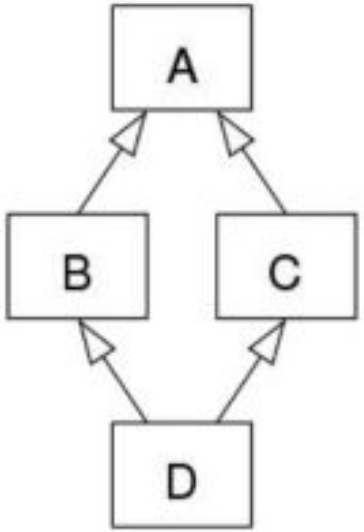
```
Shape a = new Shape();
Square b = a;
```

```
Shape a = new Square();
Square b = a;
```

- Left: allowed
- Middle: not allowed
  - `Shape` does not have the same methods or fields as `Square`
- Right: not allowed
  - Need to cast it

Inheritance in Java

- Multiple inheritance is not allowed in Java
  - Why?



Diamond Problem

Interface

- Defines what a class must be able to do, not how to do it
  - Can’t be instantiated, should only be implemented by classes
- One class can implement multiple interfaces

```
interface Vehicle {
  public void increaseSpeed();
  public void decreaseSpeed();
  public void turnLeft();
  public void turnRight();
}
```

```
class Car implements Vehicle {
  public void increaseSpeed() {
    /* Press accelerator */
  }
  public void decreaseSpeed() {
    /* Press brake pedal */
  }
  /* other implementations */
}
```

Abstract Classes

- Combination of a class and an interface
  - Similar to abstract class in C++ (pure virtual function)
- Objects of an abstract class cannot be created, can only be inherited
- `abstract` method: no implementation, must be implemented by subclasses

```
abstract class Shape {
    abstract void draw();
    void setColor(Color c) {
        /* set color */
    }
}
```

Access Modifiers

- Controls who can access an object's methods/fields
  - In general, start with `private` and make fields more visible only when necessary
- Classes also have access modifiers: `public` or no modifier (makes it package private)
- Package-private: can access within package but not outside
  - Related Java files are typically grouped together into a package

Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Thank You