Programming Fundamentals (CIS 1200) Review

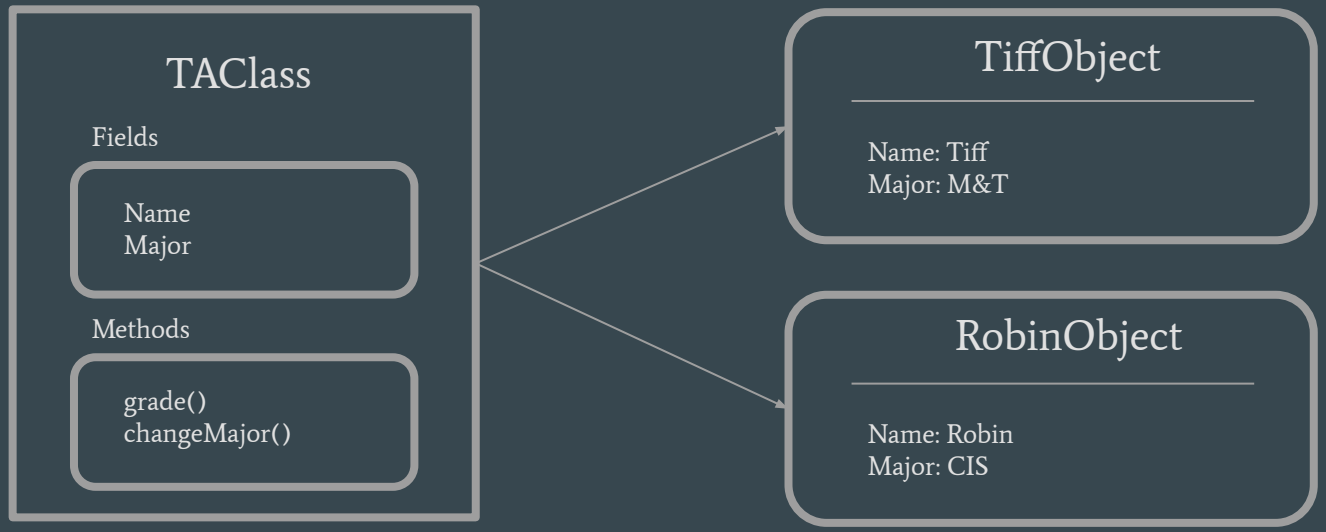
CIS 1210 Spring 2025

Agenda

- Principles of Object-Oriented Programming
  - Classes vs Objects and Inheritance
  - Abstraction
  - Encapsulation
  - Polymorphism
- Java Fundamentals
  - Primitives and Pointers
  - Static vs Dynamic Typing
  - Parameterization
- Recursion
- Common Issues

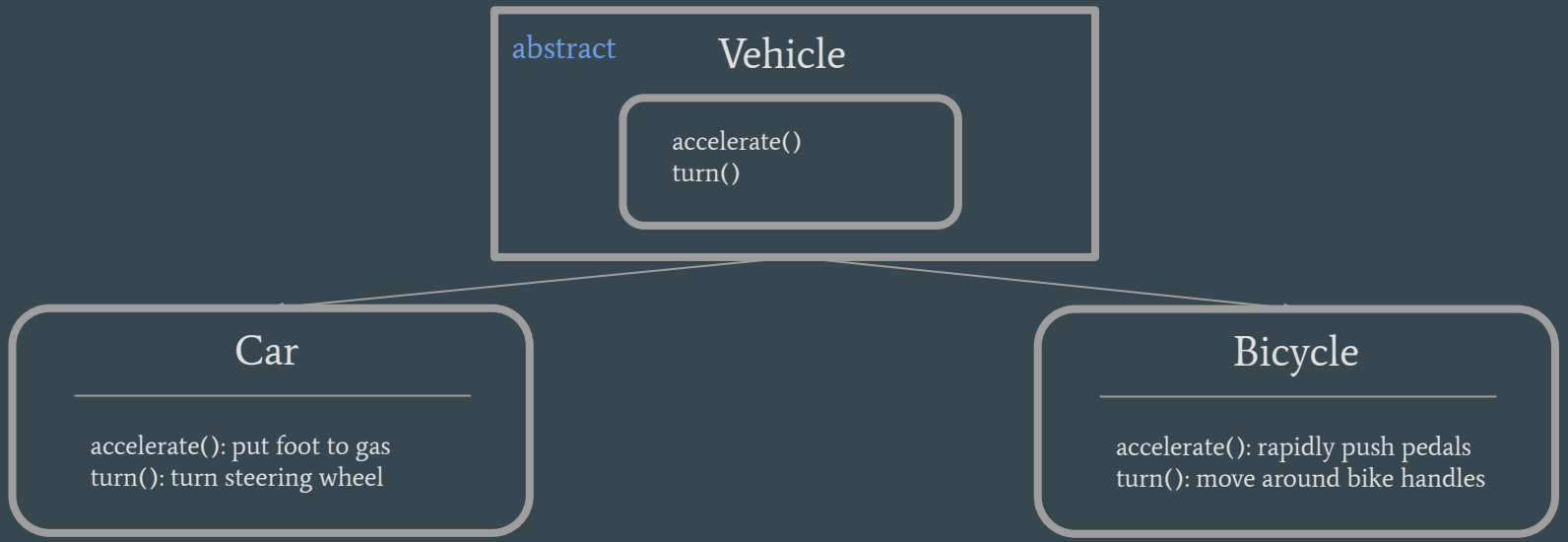
Classes vs Objects

- Classes are **templates** , objects are **instances** of classes.
- Classes can belong to **superclasses** , from which they **inherit** characteristics.



Abstraction

- Hides implementation details and only shows functionality
  - Simplifies code logic
  - Reduces programming complexity



Encapsulation

- Making methods and variables **public** or **private** will change its scope of access
  - public methods/variables can be utilized by other classes
    - Getters/setters allows control over how fields are changed
  - private methods/variables will be used solely by its root class
    - Protects fields from being unauthorized altered by client classes.

Least Restrictive ↕ Most Restrictive

Modifier	Class	Package	Subclass	World
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
package private (no modifier)	Yes	Yes	No	No
private	Yes	No	No	No

Polymorphism

- One method name can invoke different method behaviors

Overloading

Multiple methods with *same name* but *different signatures*

powerOf(int x, int y)

powerOf(double x, double y)

Overriding

Subclass's method with *same name* as superclass's method overrides it

vehicle.accelerate()

bicycle.accelerate()

Pointers and Primitives

```
Node curr = new Node(1);
setValToTwo(curr);
print(curr.value);
```

console: ‘2’

vs

```
int curr = 1;
setValToTwo(curr);
print(curr);
```

console: ‘1’

Static vs Dynamic Typing

- **Statically Typed** : type checked during compile-time (pre-execution)
  - Java's compiler will complain if an object's static type or its superclasses are an expected type.
  - *Pro*: easier to catch bugs, limits runtime errors
  - *Con*: pain to program in
  - C, Java, Haskell
- **Dynamically Typed** : type checked during run-time (during execution)
  - Will throw a runtime error if it detects an object with an unexpected type or an undefined method name
  - *Pro*: sweet to program in
  - *Con*: difficult to debug, prone to runtime errors
  - Python, JavaScript, PHP

Static Variables and Methods

- Static variables: shared between *all instances* of an object
  - If one instance updates a static field, all instances feel the effect
  - Modifying static global variables is generally bad
    - Complicates logic
    - Introduces concurrency issues
    - Common cause of inexplicably failing test cases -- don't forget to reset!

- Static methods are similar
  - Can be called without an instance of the class
    - Math.random(), Collections.sort()
  - Can not reference/modify non-static fields or call non-static methods

Parameterization

- Can parameterize classes to make more generic
- Ex: List<E>
  - Rather than creating separate classes for IntList, StringList, DoubleList, etc.
- E stands for an arbitrary type
  - Good because it's generic
  - Bad because it's you can't assume anything about it (except that it is an Object)
  - A specific type will be provided upon instantiation
    - e.g. List<Integer> list = new LinkedList<Integer>();
  - In this case, everywhere you see "E", replace with "Integer" to understand its behavior

Recursion

```
factorial(n):
    return n * factorial(n - 1)
```

```
factorial(4):
    return 4 * factorial(3)
```

```
factorial(3):
    return 3 * factorial(2)
```

```
factorial(2):
    return 2 * factorial(1)
```

```
factorial(1):
    return 1
```

Recursion

```
factorial(n):
    return n * factorial(n - 1)
```

Inductive Hypothesis

```
factorial(4):
    return 4 * factorial(3)
```

Inductive Step

```
factorial(3):
    return 3 * factorial(2)
```

```
factorial(2):
    return 2 * factorial(1)
```

Base Case

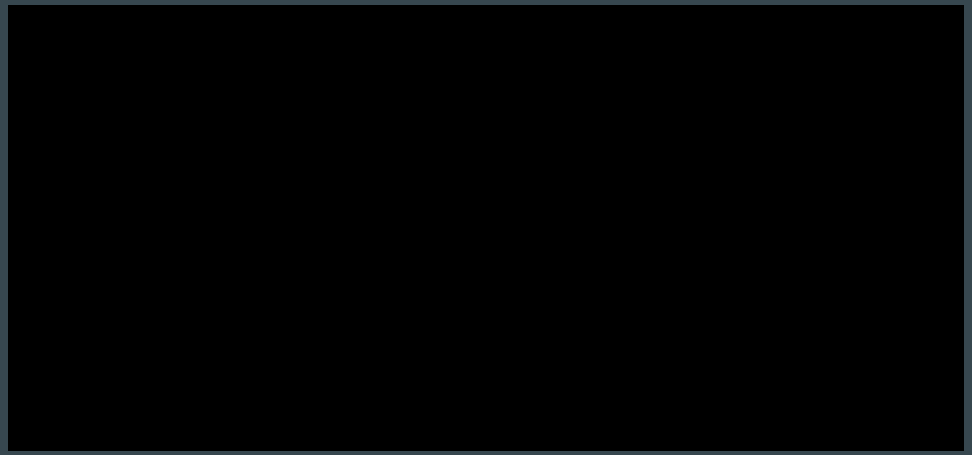
```
factorial(1):
    return 1
```

recursion = induction

Recursion

```
factorial(n):
    return n * factorial(n - 1)
```

```
factorial(4):
    return 4 * factorial(3)
```



Common Issues

- You can't instantiate an interface
  - Interfaces describe methods
  - Classes implement those methods
- Recursive functions need base cases
  - If you get a stack overflow error, you probably forgot a base case (or your program missed it)
  - Recursion is ***powerful*** but requires a "leap of faith" in the recursive step
- Objects can be null, primitives can't
  - Watch out for NPE's
  - Make sure you are testing equality correctly
  - Careful with pointers!
- Test case exhaustiveness directly correlates with better functioning program

Advice

- CIS 1210 is a little different than 1200
- Small stylistic things - make sure to name things in ways that you can remember, don't leave commented code, write good comments for yourself
- Before starting, think about how to implement things! Plan it out! Don't just start coding.
  - When are good times to save or store things?
  - What kind of data structures do you want to use? How will these change your runtime?
- Plan out test cases
  - Write out a few edge ones beforehand
- Use the debugger