

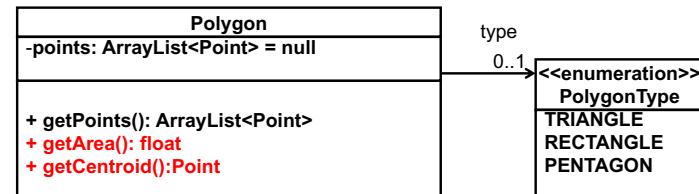
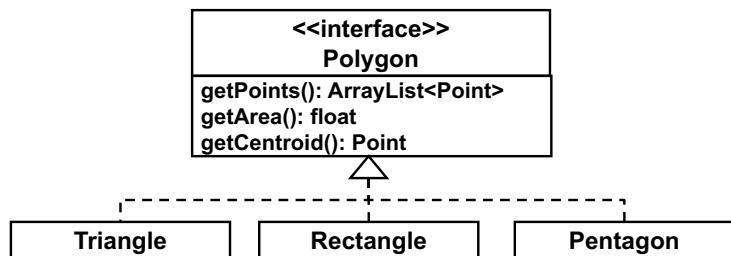
Strategy Design Pattern

- Intent
 - Define a family of algorithms
 - Encapsulate each algorithm in a class
 - Make them interchangeable
- a.k.a
 - Policy

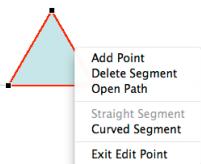
2

Strategy Design Pattern

Recap



- Can a triangle become a rectangle dynamically?
- If we allow that, eliminate class inheritance

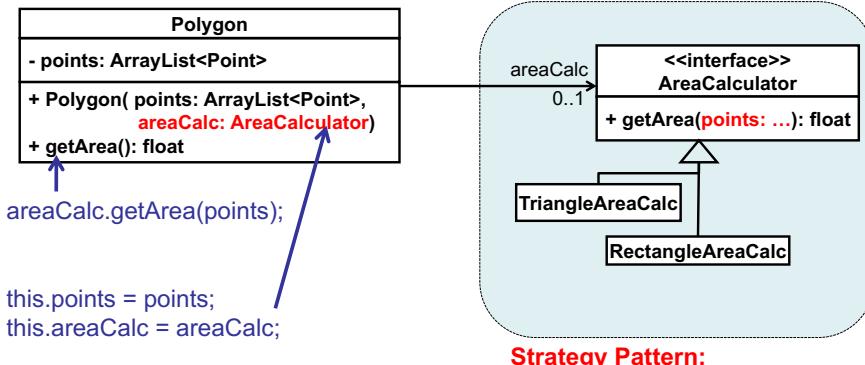


3

- Need to expect a conditional block, potentially a long and complicated one.

4

An Example of Strategy

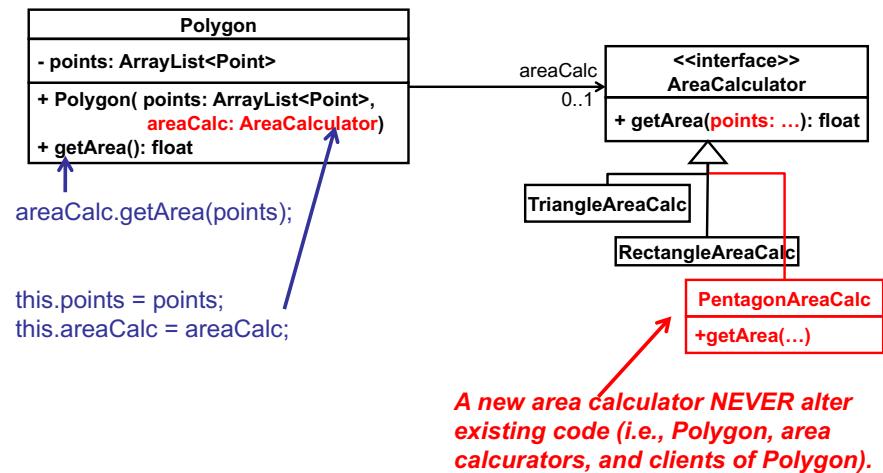


User/client of Polygon:

```

ArrayList<Point> al = new ArrayList<Point>();
al.add( new Point(...) ); al.add( new Point(...) ); al.add( new Point(...) );
Polygon p = new Polygon( al, new TriangleAreaCalc() );
p.getArea();
  
```

5



A new area calculator NEVER alter existing code (i.e., Polygon, area calculators, and clients of Polygon).

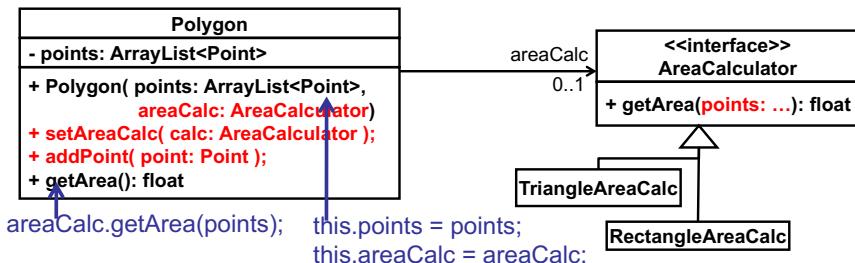
User/client of Polygon:

```

ArrayList<Point> al = new ArrayList<Point>();
al.add( new Point(...) ); al.add( new Point(...) ); .....
Polygon p = new Polygon( al, new PentagonAreaCalc() );
p.getArea();
  
```

6

Polygon Transformation



User/client of Polygon:

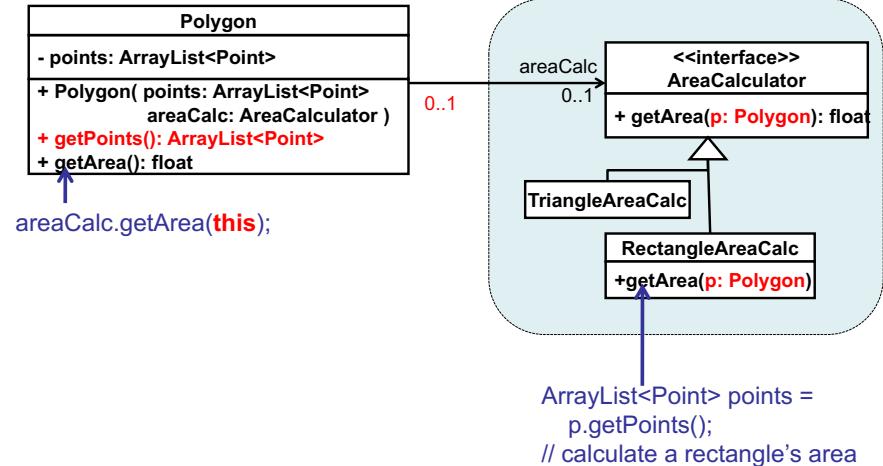
```

ArrayList<Point> al = new ArrayList<Point>();
al.add( new Point(...) ); al.add( new Point(...) ); al.add( new Point(...) );
Polygon p = new Polygon( al, new TriangleAreaCalc() );
p.getArea(); // triangle's area
p.addPoint( new Point(...) );
p.setAreaCalc( new RectangleAreaCalc() );
p.getArea(); // rectangle's area
  
```

No changes in existing code. Dynamic polygon transformation.

Dynamic replacement of area calculators

An Alternative

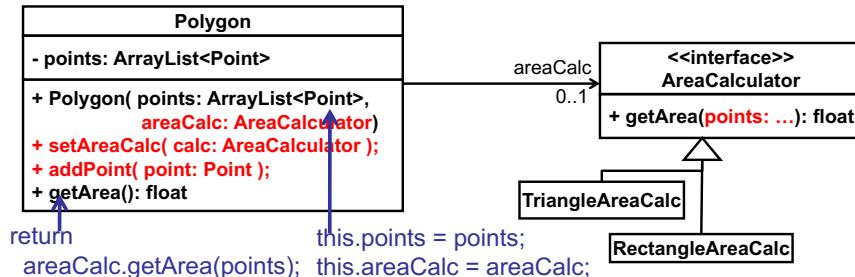


ArrayList<Point> points =
p.getPoints();
// calculate a rectangle's area

7

8

HW 5: Implement this



User/client of Polygon:

```

ArrayList<Point> al = new ArrayList<Point>();
al.add( new Point(...) ); al.add( new Point(...) ); al.add( new Point(...) );
Polygon p = new Polygon( al, new TriangleAreaCalc() );
p.getArea(); // triangle's area
p.addPoint( new Point(...) );
p.setAreaCalc( new RectangleAreaCalc() );
p.getArea(); // rectangle's area
    
```

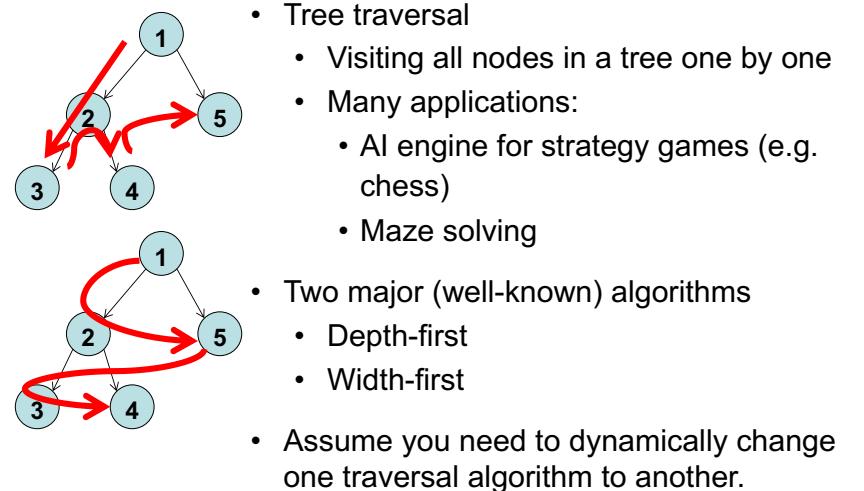
9

- Define two classes that implements the AreaCalculator interface (Triangle and Rectangle).
 - You can reuse Point in Java API or define your own.
- Implement getArea() in the two subclasses.
 - Use Heron's formula to compute a triangle's area.
 - The area of a triangle = $\text{Sqrt}(s(s-a)(s-b)(s-c))$
 - where $s=(a+b+c)/2$
 - a , b and c are the lengths of the triangle's sides.
- Write test code (in main() or with JUnit) that
 - Makes two different triangles and two different rectangles,
 - Contains those 4 polygons in a collection (e.g. ArrayList),
 - Use generics and an iterator
 - Prints out each polygon's area.
 - Take advantage of polymorphism.

10

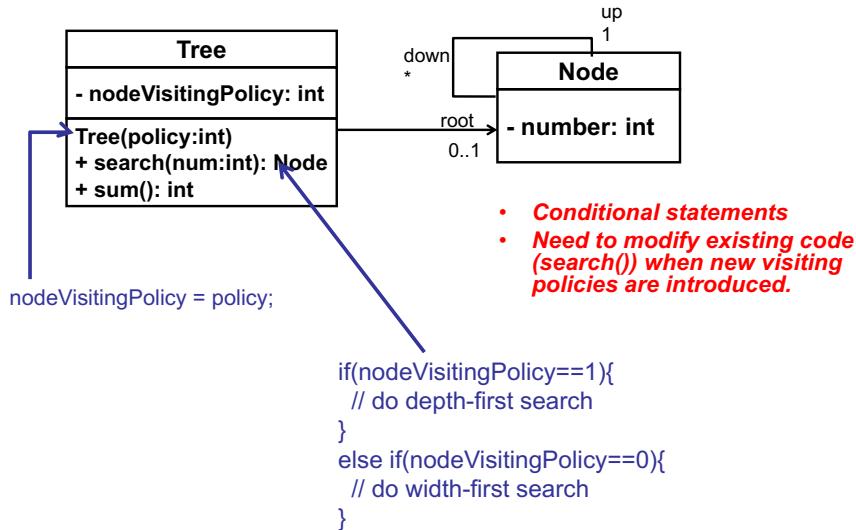
- Dynamically transform a triangle to a rectangle, and vice versa.
 - You can make a simple assumption about how the triangle and rectangle look like.
 - e.g. $[(0,0), (0,10), (10,0)] \rightarrow [(0,0), (0,10), (10,10), (10,0)]$
 - $[(0,0), (0,10), (10,10), (10,0)] \rightarrow [(0,0), (0,10), (10,0)]$
- FIRM DUE:** March 20 (Tue) midnight

Tree Traversal with Strategy

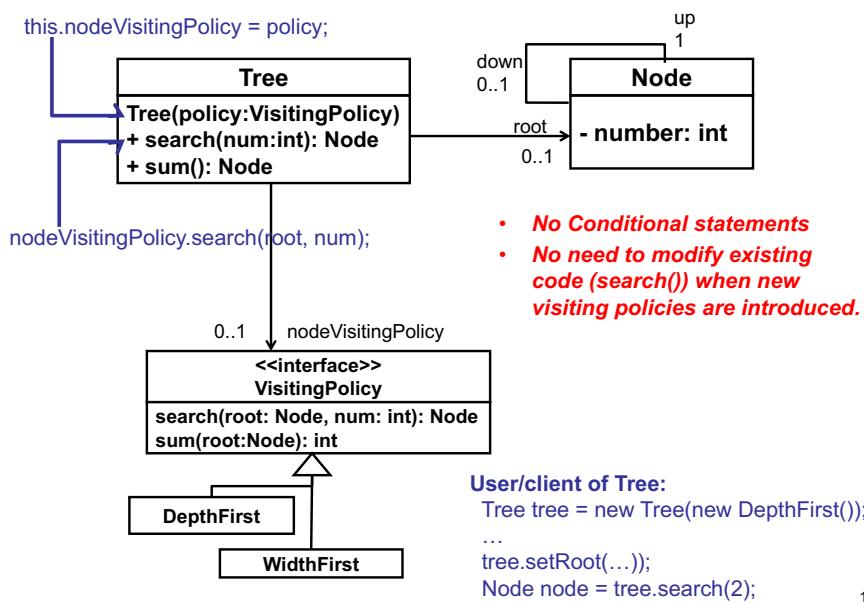
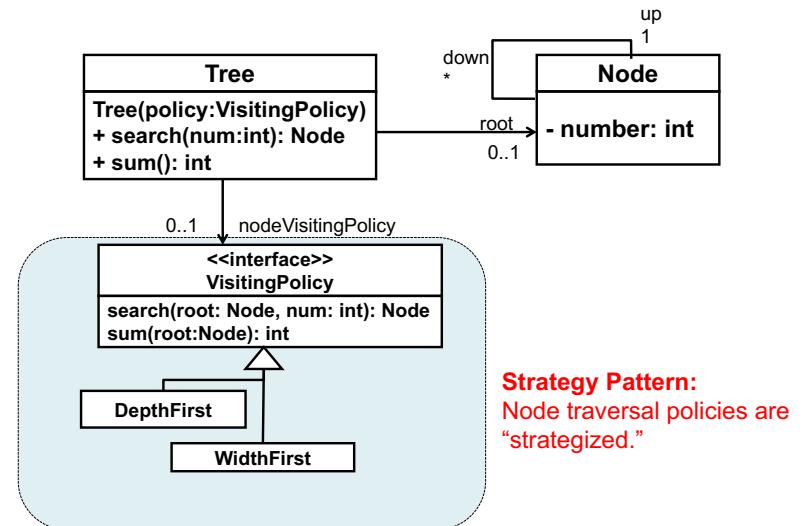


11

Not Good

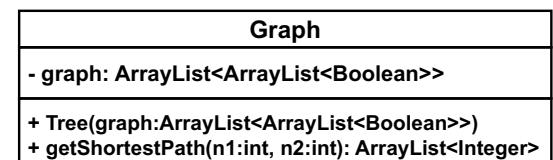
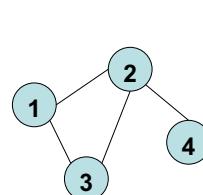


With Strategy Classes...



Graph Traversal with Strategy

- A graph consists of nodes and links.
- Requirement: Find the shortest path between given two nodes.
 - 1 → 2 → 4: 2 hops between Node 1 and Node 4
 - 2 → 3: 1 hop between Node 2 and Node 3



• 1 1 1 0
 1 1 1 1
 1 1 1 0
 0 1 0 1

Connectivity matrix

Trip Planner at mbta.org

Trip Planner
Enter an address, intersection, station or landmark below and we'll supply the best travel routes for you. [Need help?](#)

Start: South station
End: Central station

Depart at: 4:45 PM on 9/28/2010
Minimize Time and use all services
with a walking distance of 1/2 mile
Trip must be accessible
Reverse Trip Clear Search Display Trip

Itinerary 1 - Approx. 12 mins. Itinerary 2 - Approx. 12 mins.
Print Itineraries Take Red Line - Alewife To Central Sq - Outbound [view route](#).
Approx. 4:48 PM Depart from South Station - Inbound
Approx. 5:00 PM Arrive at Central Sq - Outbound

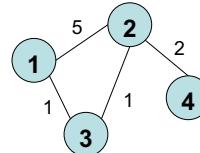
Cost:
Regular fare Senior/Disabled fare
\$2.00 \$0.60

- Directions from one place to another via T (e.g. South Station to Central Sq.)
- This is a shortest path search problem.

17

Weighted Graphs

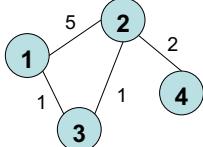
- What if you need to consider the *weighted* shortest path between two nodes?
 - 1 -> 3-> 2-> 4: total weight = 4, between Node 1 and Node 4
 - 1 -> 3 -> 2: total weight = 2, between Node 1 and Node 2



Graph
- graph: ArrayList<ArrayList<Integer>>
Tree(graph:ArrayList<ArrayList<Integer>>) + getShortestPath(n1:int, n2:int): ArrayList<Integer>

- 0 5 1 -1
5 0 1 2
1 1 0 -1
-1 2 -1 0
- Weighted connectivity matrix

18

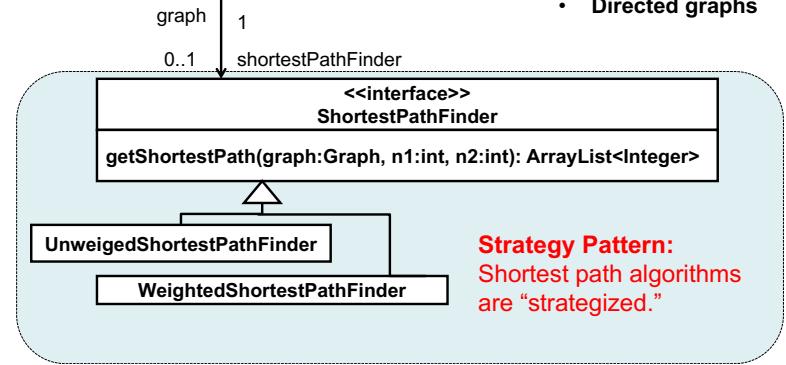


Graph
- graph: ArrayList<ArrayList<Boolean>>
Tree(graph:ArrayList<ArrayList<Boolean>>) + getShortestPath(n1:int, n2:int): ArrayList<Integer>

- Add conditional statements in getShortestPath()
 - Conditional statements
 - Need to modify existing code when new algorithms are introduced to compute the shortest path.
- Add getWeightedShortestPath(...)
 - No conditional statements
 - Still need to modify existing code when new algorithms are introduced to compute the shortest path.

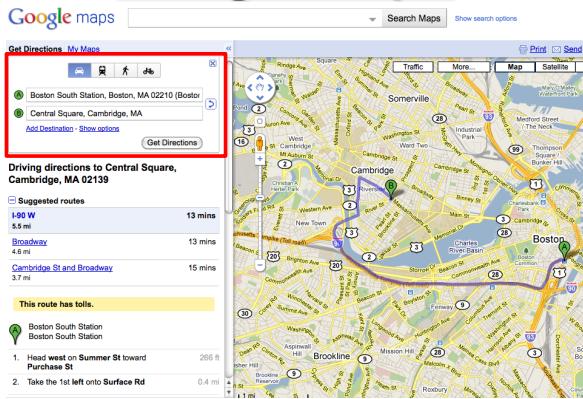
19

Graph
- graph: ArrayList<ArrayList<Integer>>
Tree(graph:ArrayList<ArrayList<Integer>>) + getShortestPath(n1:int, n2:int): ArrayList<Integer>



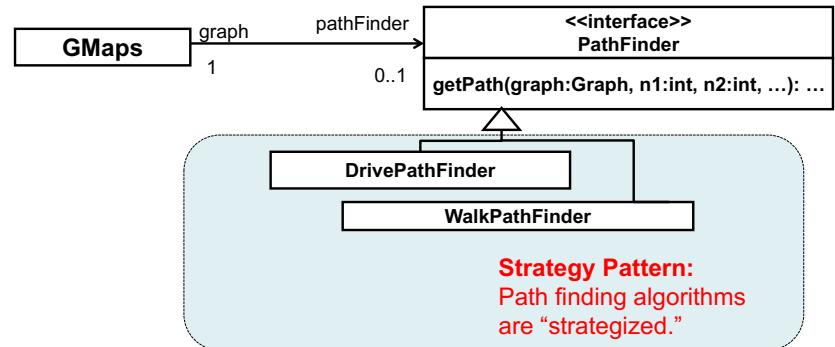
20

Google Maps



- Directions from one place to another (e.g. South Station to Central Sq.)
 - By car
 - By T
 - By walk
 - By bicycle
 - ... extra transportation means in the future? With Uber and Lyft.

21



22

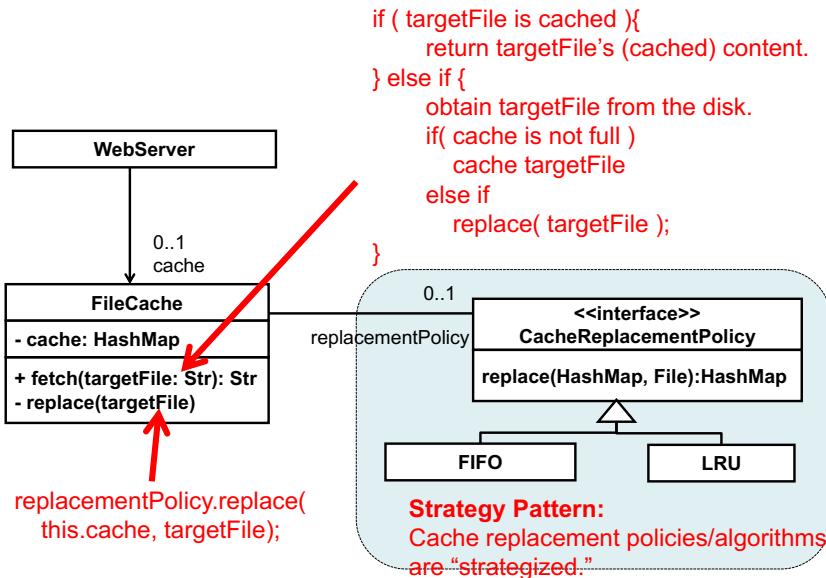
Web Page Caching

- A web server
 - Receives a connection establishment request from a client (browser).
 - Receives an HTTP command
 - Retrieves a target HTML file from the local disk and returns it to the client.
 - May **cache** a set of HTML files that have been accessed in the past.
 - Keeps them in the memory (e.g., with `HashMap`)
 - Returns a cached file to the client from the next time.
 - Benefit: Faster response to the clients
 - » Memory access is much faster than disk access.

Cache Replacement

- Issue: **Cache size** is limited.
 - Memory size is limited.
 - Need to decide **which cached file to be removed** so that a new file can be cached
 - When the number/size of cached files reaches the maximum cache size.
- Various cache replacement policies/algorithms exist.
 - FIFO (First In First Out)
 - LRU (Least Recently Used)
 - LFU (Least Frequently Used)
 - ... etc.

25



Comparators

- Sorting array elements:

```

int years[] = {2010, 2000, 1997, 2006};
Arrays.sort(years);
for(int y: years)
    System.out.println(y);

```

- `java.util.Arrays`: a utility class (a collection of static methods) to process arrays and array elements
- `sort()` sorts array elements in an ascending order.
 - 1997 -> 2000 -> 2006 -> 2010

26

27

- Sorting collection elements:

```

ArrayList<Integer> years2 = new ArrayList<Integer>();
years2.add(new Integer(2010));
years2.add(new Integer(2000));
years2.add(new Integer(1997));
years2.add(new Integer(2006));
Collections.sort(years2);
for(Integer y: years2)
    System.out.println(y);

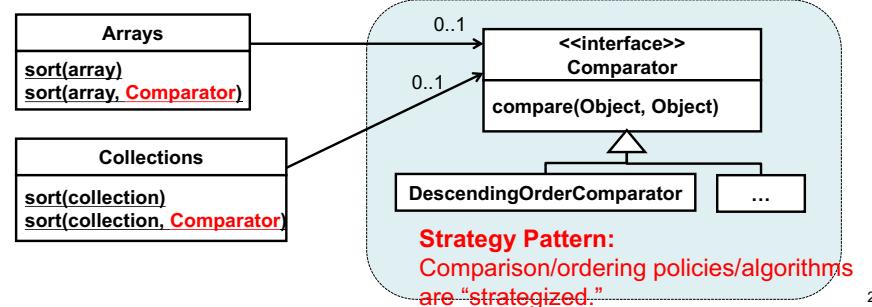
```

- `java.util.Collections`: a utility class (a collection of static methods) to process collections and collection elements
- `sort()` sorts array elements in an ascending order.
 - 1997 -> 2000 -> 2006 -> 2010

28

Comparison/Ordering Policies

- What if you want to sort array/collection elements in a descending order or any specialized (user-defined) order?
 - `Arrays.sort()` and `Collections.sort()` implement ascending ordering only.
 - They do not implement any other policies.
- Define a custom comparator by implementing `java.util.Comparator`



29

Sorting Collection Elements with a Custom Comparator

- `Arrays.sort()` and `Collections.sort()` are defined to sort array/collection elements from “smaller” to “bigger” elements.
 - By default, “smaller” elements mean the elements that have *lower* numbers.
- A descending ordering can be implemented by treating “smaller” elements as the elements that have *higher* numbers.
- `compare()` in comparator classes can define (or re-define) what “small” means and what’s “big” means.
 - Returns a negative integer, zero, or a positive integer as the first argument is “smaller” than, “equal to,” or “bigger” than the second.

```
• public class DescendingOrderComparator implements Comparator{  
    public int compare(Object o1, Object o2){  
        return ((Integer)o2).intValue() - ((Integer) o1).intValue();  
    }  
}
```

30

```
- ArrayList<Integer> years = new ArrayList<Integer>();  
years.add(new Integer(2010)); years.add(new Integer(2000));  
years.add(new Integer(1997)); years.add(new Integer(2006));  
Collections.sort(years);  
for(Integer y: years)  
    System.out.println(y);  
Collections.sort(years, new DescendingOrderComparator());  
for(Integer y: years)  
    System.out.println(y);
```

- 1997 -> 2000 -> 2006 -> 2010
- 2010 -> 2006 -> 2000 -> 1997

```
• public class DescendingOrderComparator implements Comparator{  
    public int compare(Object o1, Object o2){  
        return ((Integer)o2).intValue() - ((Integer) o1).intValue();  
    }  
}
```

- A more type-safe option is available/recommended:

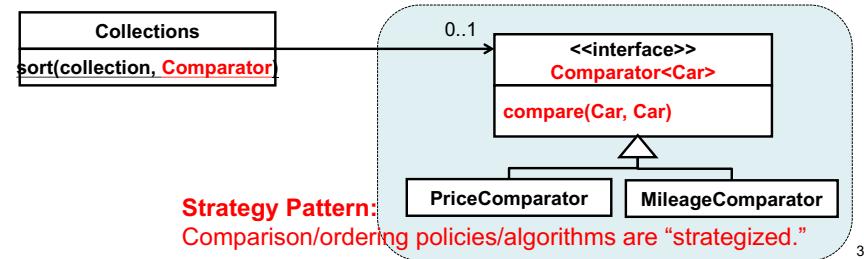
```
• public class DescendingOrderComparator<Integer>{  
    implements Comparator<Integer>{  
        public int compare(Integer o1, Integer o2){  
            return o2.intValue() - o1.intValue();  
        }  
    }  
}
```

32

- What if you want to sort a collection of your own (i.e., user-defined) objects?

```
- class Car{  
    public int getPrice();  
    public int getYear();  
    public float getMileage();  
}  
- ArrayList<Car> usedCars= new ArrayList<Car>();  
usedCars.add(new Car(...)); usedCars.add(...); usedCars.add(...);  
Collections.sort(usedCars);
```

- Need to define a car-ordering policy as a custom comparator class.



31

33

Thanks to Strategy...

- Assume “bigger” (or better) elements as the elements that have a
 - Lower price
 - Higher (more recent) year
 - Lower mileage

```

public class PriceComparator<Car>
    implements Comparator<Car>{
    public int compare(Car car1, Car car2){
        return car2.getPrice() - car1.getPrice();
    }
}

public class YearComparator<Car>
    implements Comparator<Car>{
    public int compare(Car car1, Car car2){
        return car1.getYear() - car2.getYear();
    }
}
  
```

34

- You can define any extra ordering/comparison policies without changing existing code
 - e.g., `Car.Collections.sort()`
 - No conditional statements to shift ordering/comparison policies.

- You can dynamically change one ordering/comparison policy to another.

```

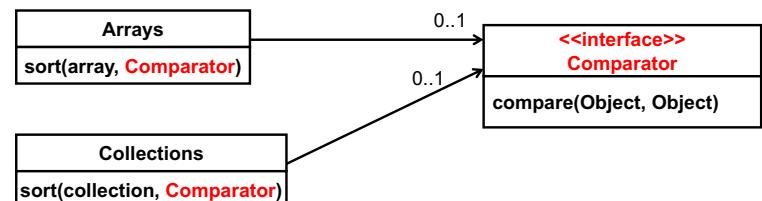
Collections.sort(usedCars, new PriceComparator());
// printing a list of cars
Collection.sort(usedCars, new YearComparator());
// printing a list of cars
  
```

35

Used Car Listings

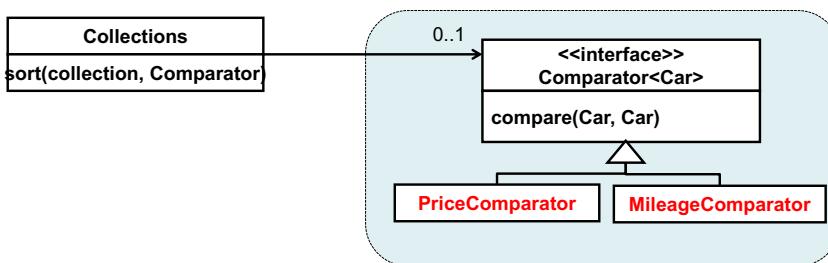
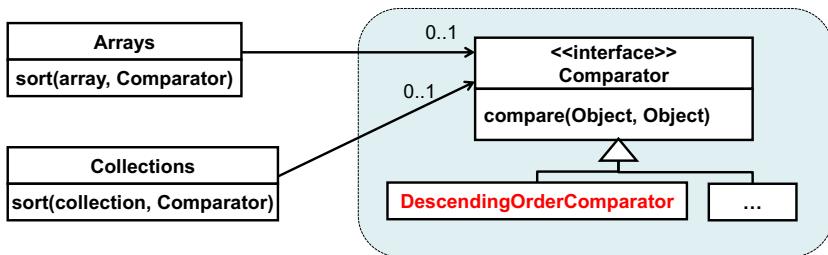
Year/Model	Information	Mileage	Seller/Distance	Price
2000 Audi A4 5dr Wgn 1.8T Avant Auto Quattro AWD	Used MPG: 19 City / 28 Hwy Automatic Gray	136,636	Dedham Auto Mall (7.4 Miles) Search Dealer Inventory	\$4,880
			 Free CARFAX Report	
2001 Audi A4	Used	84,297	Herb Connolly Hyundai (18.8 Miles) Search Dealer Inventory	\$7,995
			 Free CARFAX Report	
2002 Audi A6 4dr Sdn quattro AWD Auto	Used MPG: 17 City / 25 Hwy Automatic Blue	84,272	Dedham Auto Mall (14 Miles) Search Dealer Inventory	\$7,998
			 Free CARFAX Report	
2003 Audi A4 1.8T	Used MPG: 20 City / 28 Hwy Automatic Blue	78,321	Direct Auto Mall (18.8 Miles) Search Dealer Inventory	\$10,687
			 Get a CARFAX Record Check	
2002 Audi allroad 5dr quattro AWD Auto	Used MPG: 15 City / 21 Hwy Automatic Green	98,382	Lux Auto Plus (8.6 Miles) Search Dealer Inventory	\$10,900
			 Get a CARFAX Record Check	
2008 Audi A6	Certified Pre-Owned MPG: 17 City / 25 Hwy Automatic	0	Audi Burlington & Porsche of Burlington (14.9 Miles) Search Dealer Inventory	\$37,897
			 Get a CARFAX Record Check	
2007 Audi A4	Used MPG: 22 City / NA Hwy Brilliant Black	6,822	(19.3 Miles)	\$24,995
			 Get a CARFAX Record Check	
2009 Audi A4	Certified Pre-Owned White	10,120	Audi Burlington & Porsche of Burlington (14.9 Miles) Search Dealer Inventory	\$33,497
			 Get a CARFAX Record Check	
2009 Audi A4 3.2L Prestige	Certified Pre-Owned MPG: 17 City / 26 Hwy Automatic White	12,118	Audi Burlington & Porsche of Burlington (14.9 Miles) Search Dealer Inventory	\$39,877
			 Get a CARFAX Record Check	
2008 Audi S5	Used Brilliant Black	16,492	(19.3 Miles)	\$44,995
			 Get a CARFAX Record Check	

Programming to an Interface



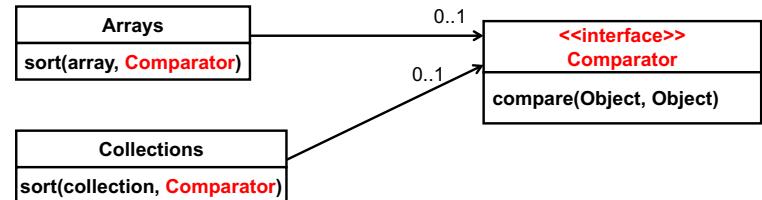
- One of the most important points in *Strategy*
- What Java API designers did was to...
 - Define the interface `Comparator` only.
 - Users of `Arrays/Collections` will define their comparator implementations.

37



38

Programming to an Interface



- What Java API designers intended was to...
 - make Arrays.sort() and Collections.sort() intact even if changes occur in Comparator implementations
 - make Arrays and Collections loosely-coupled from Comparator implementations.

39

HW 6

- Step 1: Implement the Car class and three comparator classes
- Step 2: Implement an extra comparator class, `ParetoComparator<Car>`, which performs the *Pareto comparison*.
- Test cases make several cars and sort them with four comparators.
- Due: March 29 (Thu) midnight

40

Pareto Comparison

- Given multiple objectives,
 - e.g., price, year and mileage
 - Car A is said to **dominate** (or outperform) Car B iff:
 - A's objective values are superior than, or equal to, B's in all objectives, and
 - A's objective values are superior than B's in at least one objective.
- Count the number of cars that dominate each car.
- | Car | Count |
|-----|-------------------------|
| A | 0 (No cars dominate A.) |
| B | 3 (A, D, E) |
| C | 4 (A, B, D, E) |
| D | 1 (E) |
| E | 0 (No cars dominate E.) |
- Better cars have lower “domination counts.”
- To order cars from the best one(s) to the worst one(s), compare() should treat “better” ones as “smaller” ones.
-
- The scatter plot shows five cars labeled A through E. The y-axis is labeled 'mileage' and the x-axis is labeled 'price'. The points are arranged such that A is the lowest left point, B is above and to the right of A, C is above and to the right of B, D is below and to the right of B, and E is the highest right point. Dashed lines connect each point to its respective axis labels.

41

- Implement getDominationCount() in Car.
- When to compute domination counts for individual cars?
 - Before calling sort()

```

• // finish up computing domination counts for all cars
  // by calling getDominationCount() on those cars, and
  // then call sort()
  Collections.sort(usedCars, new DominationComparator<Car>());

```

- When calling sort()

```

• // DominationComparator's constructor calls
  // getDominationCount() on individual cars.
  Collections.sort(usedCars,
    new DominationComparator<Car>(usedCars));

```

42

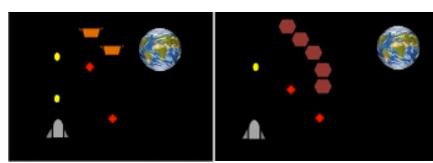
Suggested Read

- Replace Type Code with Class (incl. enumeration)
 - <http://sourcemaking.com/refactoring/replace-type-code-with-class>
- Replace Type Code with State/Strategy
 - <http://sourcemaking.com/refactoring/replace-type-code-with-state-strategy>
- Replace Type Code with Subclasses
 - <http://sourcemaking.com/refactoring/replace-type-code-with-subclasses>
- Replace Conditional with Polymorphism
 - <http://sourcemaking.com/refactoring/replace-conditional-with-polymorphism>

43

One More Exercise

Imagine a Simple 2D Shooting Game



0: rectangle
1: hexagon

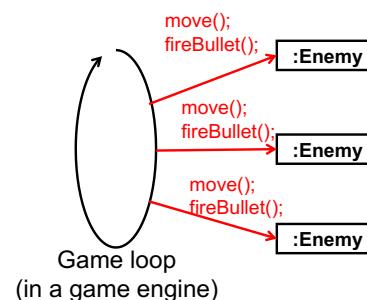
Enemy
- enemyType: int
- location: Point
- numBullets: int
+ move(): void
+ fireBullet(): void

```

switch(enemyType){
  case 0:
    ...
    break;
  case 1:
    ...
  switch(enemyType){
    case 0:
      ...
      break;
    case 1:
      ...
  }
}

```

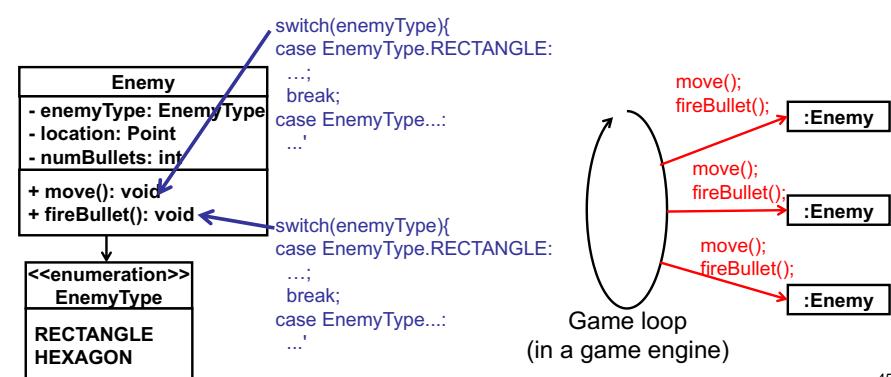
- Each type of enemies has its own attack pattern.
 - e.g. How to move, when to fire bullets, how to fire bullets...



44

What's Bad?

- Using magic numbers.
 - Replace them with symbolic constants or an enumeration.
 - c.f. Lecture note #3



45

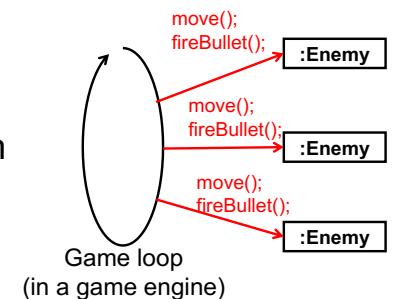
Still Not Good

- Conditional blocks. Error-prone to maintain them
 - If there are many enemy types.
 - If new enemy types may be added in the near future.
 - Imagine 3,000 or 5,000 lines of code for each conditional block
 - If repetitive conditional blocks exist.
- Attack patterns (moving patterns and firing patterns) are tightly coupled with Enemy. Hard to maintain them
 - If attack patterns often change.
 - Revising hexagonal enemy's moving pattern
 - Having the same moving pattern for rectangle and hexagonal enemies.
 - Changing rectangle enemy's moving pattern to be more intelligent as you play
 - Introducing a new type of enemies and having them use hexagonal enemy's moving pattern
 - Introducing a new type of enemies and implementing a new pattern for them.

46

What We Want are to...

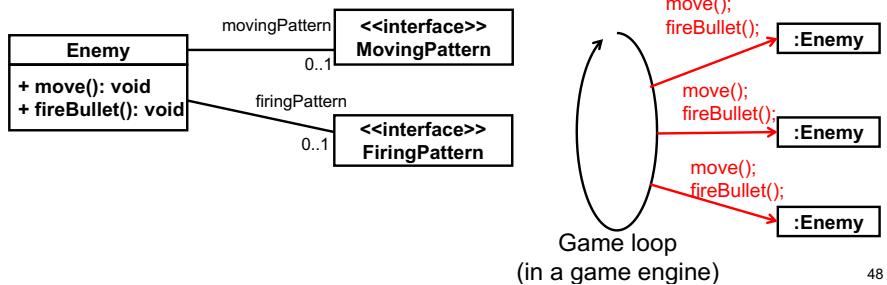
- Eliminate those conditional blocks.
- Separate Enemy and its attack patterns (moving patterns and firing patterns).
 - Make Enemy and its attack patterns loosely coupled.
- Define a family of attack patterns (algorithms) in a unified way
- Encapsulate each algorithm in a class
- Make algorithms interchangeable



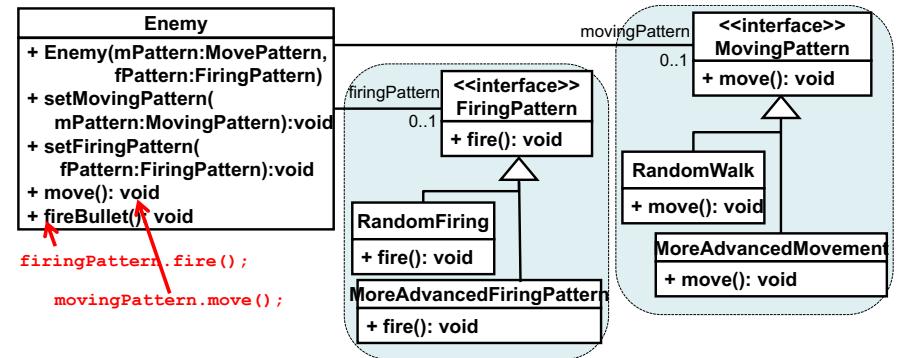
47

Complete the Design with Strategy

- Complete the class diagram with *Strategy*
 - Add classes, methods and data fields as you like.
- Show how Enemy's move() and fireBullet() look like.

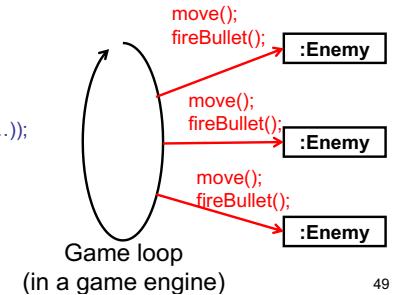


48



User/client of Enemy:

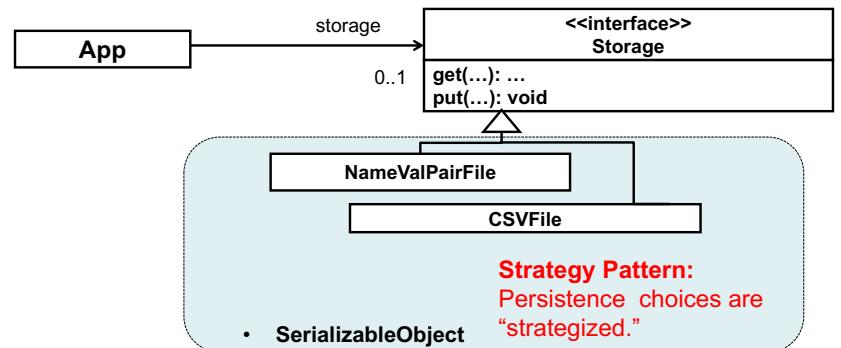
```
Enemy e1 = new Enemy(new RandomWalk(...),  
                    new RandomFiring(...));  
Enemy e2 = new Enemy(new RandomWalk(...),  
                    new MoreAdvancedPattern(...));  
Enemy e3 = ...  
ArrayList<Enemy> el = new ArrayList<Enemy>();  
el.add(e1); el.add(e2); el.add(e3);  
for(Enemy e: el){  
    e.move();  
    e.fireBullet(); }
```



49

Data Storage (Persistence)

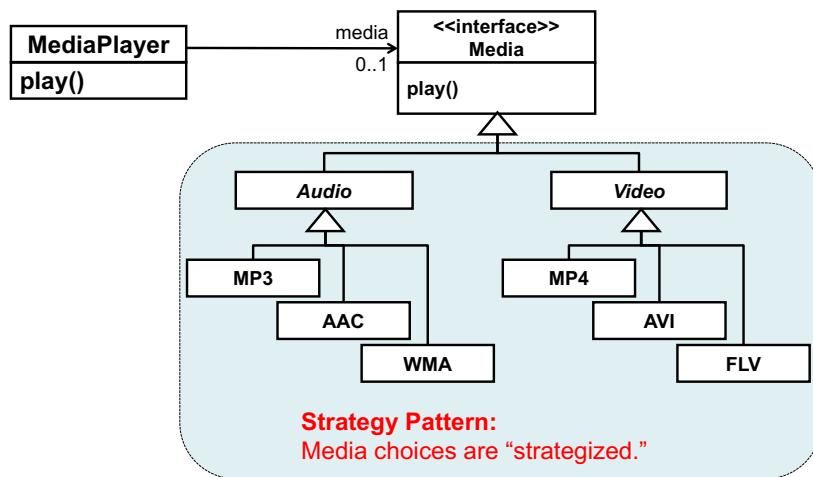
Other Examples of Strategy



50

51

Media Player



52