Inheritance

Lab 5

TA : Hyuna Seo, Kichang Yang, Minkyung Jeong, Jaeyong Kim

Announcement

- You should finish the lab practice and submit your job to eTL before the next lab class starts(Wednesday, 7:00 PM).
- The answer of the practice will be uploaded after the due.

Overview

- Recap: Inheritance review
  - Inheritance basics
  - Overriding / Super
  - Abstract class
- Problem - Evolution of Trust

Recap: Inheritance Basics

```
class Lecture {
    boolean hasTA = true;
    boolean hasExams = true;
    boolean hasAssignments = true;
}
class CSLecture {
    boolean hasTA = true;
    boolean hasExams = true;
    boolean hasAssignments = true;
    boolean isHard = true;

}
class CPLecture {
    boolean hasTA = true;
    boolean hasExams = true;
    boolean hasAssignments = false;
    boolean isExciting = true;
}
```



```
class Lecture {
    boolean hasTA = true;
    boolean hasExams = true;
    boolean hasAssignments = true;
}
class CSLecture extends Lecture {
    boolean isHard = true;
}
class CPLecture extends CSLecture {
    boolean hasAssignments = false;
    boolean isExciting = true;
}
```

Recap: Overriding

```
class Parent {
    void printName() { System.out.println("Parent"); }
}
class Child extends Parent {
    @Override // It raises an error if there is no "printName" method in "Parent" class
    void printName() { System.out.println("Child"); }
}
```

main Method

```
Parent parent = new Parent();
Child child = new Child();
parent.printName();
child.printName();
```

Output

```
Parent
Child
```

Recap: Super

```
class Parent {
   void printName() { System.out.println("Parent"); }
}

class Child extends Parent {
   @Override
   void printName() { System.out.println("Child"); }
   void printParentName() { super.printName(); }
}
```

main Method

```
Child child = new Child();
child.printName();
child.printParentName();
```

Output

```
Child
Parent
```

Abstract class

```
abstract class Person {
    String description;
    String name;
    private int age;
    Person(String description, String name, int age) {
        this.description = description;
        this.name = name;
        this.age = age;
    }
    public void printName() {
        System.out.println(description + " " + name);
    }
    public void ageOneYear() {
        age++;
        System.out.printf("%s is now %d years old.\n", name, age);
    }
    abstract void work();
    abstract void play();
}
```

Child classes need to implement these on their own

Abstract class

```
class BusinessMan extends Person {
   BusinessMan(String description, String name, int age) {
       super(description, name, age);
   }
   public void work() {
       System.out.println("Meeting all day");
   }
   public void play() {
       System.out.println("Go to the movies");
   }
}

class Student extends Person {
   Student(String description, String name, int age) {
       super(description, name, age);
   }
   public void work() {
       System.out.println("Study hard");
   }
   public void play() {
       System.out.println("Drink hard");
   }
}
```

Businessman-specific code

Student-specific code

Objectives

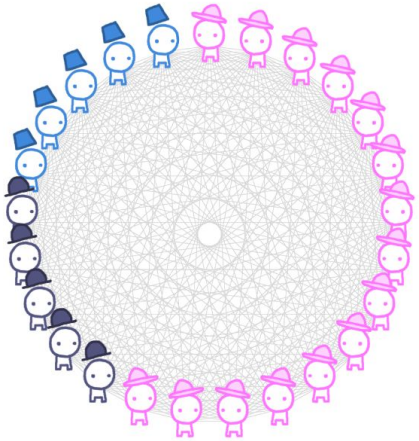
- Get used to Inheritance
- Problem - Evolution of Trust
  - Inheritance basics
  - Overriding / Super
  - Abstract class

Problem Overview

- Problem - Evolution of Trust
  - Subprob 1 : The simplest game (0:05)
  - Subprob 2 : Implement agents (0:15)
    - 2-1 Angel vs. Devil
    - 2-2 Copycat
    - 2-3 Describing an agent
  - Subprob 3 : Pairwise match (0:05)
  - Subprob 4 : Evolving the population (0:10)

Problem : Evolution of Trust

- https://ncase.me/trust/
- Simulation of the prisoner's dilemma
- Let's play the game for 10 minutes to grasp the concept!



Say we start with the following population of players: 15 Always Cooperates, 5 Always Cheats, and 5 Copycats. (We'll ignore Grudger & Detective for now)

We're going to do the tournament-eliminate-reproduce dance a dozen times or so. Let's make another bet! Who do you think will win the *first* tournament? **PLACE YOUR BETS, AGAIN:**

All Cooperate

All Cheat

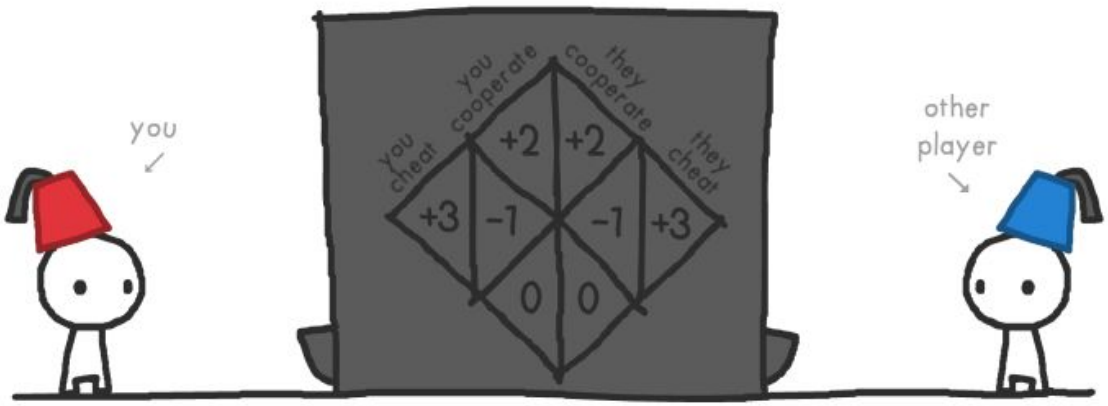
Copycat

(forgot who's who? hover buttons to see descriptions of each character!)

Problem : Evolution of Trust

- **The Game of Trust**

You have one choice. In front of you is a machine: if you put a coin in the machine, the *other player* gets three coins – and vice versa. You both can either choose to COOPERATE (put in coin), or CHEAT (don't put in coin).



- **Characters**

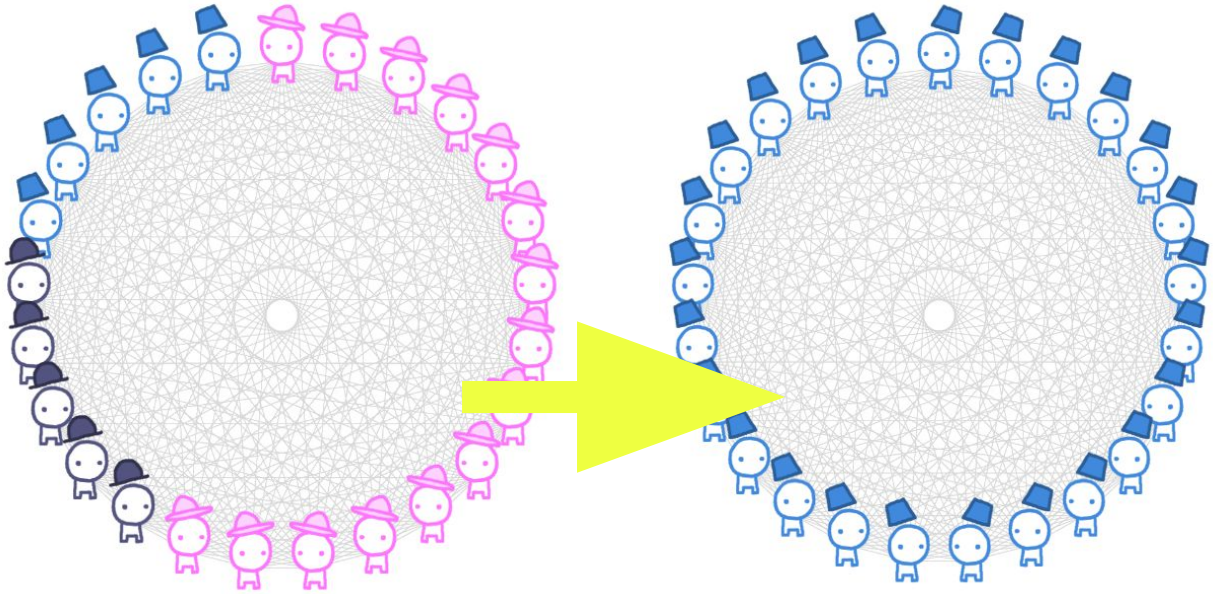
COPYCAT: Hello! I start with Cooperate, and afterwards, I just copy whatever you did in the last round. Meow

ALWAYS CHEAT: *the strong shall eat the weak*

ALWAYS COOPERATE: Let's be best friends! <3

Problem : Evolution of Trust

- Let’s demonstrate this!



...Copycat inherits the earth.

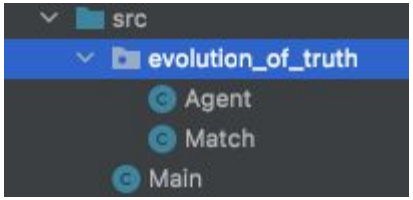
So, in the long run, you were right - Copycat wins! Always Cheat may have won in the short run, but its exploitativeness was its downfall. This reminds me of a quote:

"We are punished by our sins, not for them."
~ Elbert Hubbard

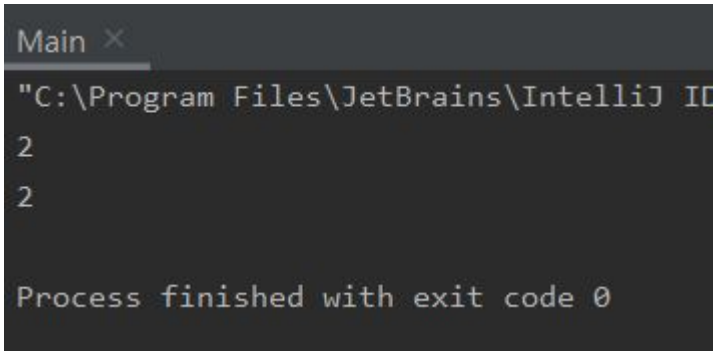
(oh, and by the way...)

Subprob 1 : The simplest game (5 min)

- Now let's make the simplest game
  - Two agents only choose to **cooperate**
- Project initialization
  - Make a package named `evolution_of_truth`
    - Inside, make two classes `Agent` and `Match`
  - Make a `Main` class at the top `src` directory
    - It is our entry point



Output



Subprob 1 : The simplest game (5 min)

evolution_of_truth.Agent.java

```
package evolution_of_truth;


public class Agent{

    private int score;

    public Agent() {
        score = 0;
    }

    public int getScore() {
        // TODO : return this agent's current score
    }

    public void setScore(int newScore) {
        // TODO : set this agent's score to the given
                  new score
    }

    public int choice(){
        return Match.COOPERATE;
    }
}
```

evolution_of_truth.Match.java

```
import evolution_of_truth;

public class Match {

    public static int CHEAT = 0;
    public static int COOPERATE = 1;
    public static int UNDEFINED = -1;

    // Sets the value each player gets for all possible cases
    private static int ruleMatrix[][][] = {
            {
                    {0, 0}, // eg) A cheats, B cheats
                    {}      // TODO : A cheats, B cooperates
            },
            {
                    {},     // TODO : A cooperates, B cheats
                    {}      // TODO : A cooperates, B cooperates
            }
    };

    public static void playGame(Agent agentA, Agent agentB) {
        // TODO : Get choices of both agents

        // TODO : Set scores of both agents according to the
result
                of the game
    }
}
```

Subprob 1 : The simplest game (5 min)

Main.java

```
import evolution_of_truth.Agent;

import evolution_of_truth.Match;

public class Main {
    public static void main(String args[]) {

            /* TODO:
                Create 2 Agents - agentA, agentB

                Play a game with agentA and agentB

                Print the score of both agents
            */

        }
    }
}
```

Subprob 2-1 : Angel vs. Devil (5 min)

- Now let's implement  and 
  - Then see if angel loses a score while devil wins
- Make a new package `agent` inside `evolution_of_truth`
  - Move `Agent` to the package
    - Say 'yes' to refactoring option
  - Make two classes `Angel` and Devil

Subprob 2-1 : Angel vs. Devil (5 min)

evolution_of_truth.agent.Angel.java

```java
package evolution_of_truth.agent;

import evolution_of_truth.Match;

public class Angel extends Agent{

    @Override
    public int choice(){
        /*
            Angel always cooperates
        */
    }
}
```

evolution_of_truth.agent.Devil.java

```java
package evolution_of_truth.agent;

import evolution_of_truth.Match;

public class Devil extends Agent{

    @Override
    public int choice(){
        /*
            Devil always cheats
        */
    }
}
```

Subprob 2-1 : Angel vs. Devil (5 min)

- Once subclasses `Angel` and `Devil` defined, the choice of `Agent` become ambiguous

- Let's make `Agent` an abstract class
  - It will serve as an outline of agents
  - It shouldn't be instantiated directly.

Subprob 2-1 : Angel vs. Devil (5 min)

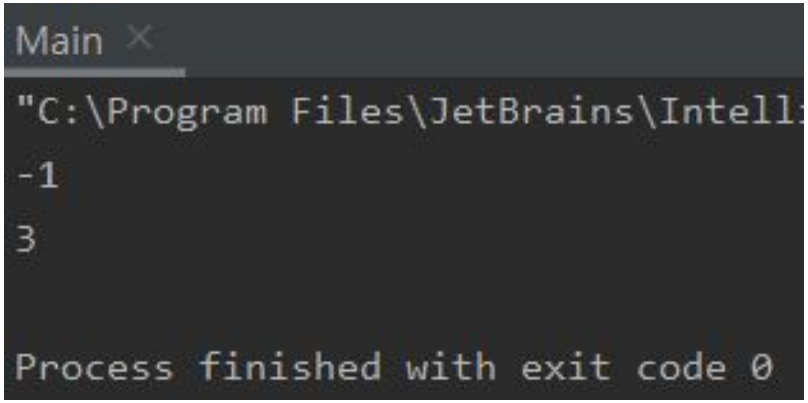
evolution_of_truth.agent.Agent.java

```
package evolution_of_truth;
package evolution_of_truth.agent;

public class Agent{
abstract public class Agent{

    private int score;

    public Agent() {
        score = 0;
    }

    public int getScore() {
        ...
    }

    public void setScore(int newScore) {
        ...
    }

    public int choice(){
        return Match.COOPERATE;
    }
    abstract public int choice();
}
```

src/Main.java

```
...  @@ -1,10 +1,12 @@
     - import evolution.of.truth.Agent;
1    + import evolution.of.truth.agent.Agent;
2      import evolution.of.truth.Match;
3    + import evolution.of.truth.agent.Angel;
4    + import evolution.of.truth.agent.Devil;
5
6      public class Main {
7          public static void main(String args[]) {
     -          Agent agentA = new Agent();
     -          Agent agentB = new Agent();
8    +          Agent agentA = new Angel();
9    +          Agent agentB = new Devil();
10             Match.playGame(agentA, agentB);
11             System.out.println(agentA.getScore());
12             System.out.println(agentB.getScore());
```

- Delete - lines from the code and add + lines



- Okay, an angel is being exploited
- Note how we didn't change anything in Match.java
  - And very little change in Main.java

Subprob 2-2 : Copycat (5 min)

- Copycat needs previous choice of the opponent

COPYCAT: Hello! I start with Cooperate, and afterwards, I just copy whatever you did in the last round. Meow

- But our `choice` function doesn't have any parameters
- Add int parameter `previousOpponentChoice` to all `choice` functions

Subprob 2-2 : Copycat (5 min)

evolution_of_truth.agent.Copycat.java

```
package evolution_of_truth.agent;

import evolution_of_truth.Match;

public class Copycat extends Agent {

    @Override
    public int choice(int previousOpponentChoice){
        /*
            Copycat starts with cooperate,
            then it simply repeats whatever the opponent did
            in the last round
        */
     }

}
```

Subprob 2-2 : Copycat (5 min)

- So where does `previousOpponentChoice` come from?
- It should be defined for every pair of agents
  - i.e., a copycat, having been cheated by a devil, shouldn't take revenge on an innocent angel

- Let's modify `Match` class!
  - From now on, we will make an instance of Match for every pair of agents

Subprob 2-2 : Copycat (5 min)

evolution_of_truth.Match.java

```
...
    Agent agentA, agentB;
    int previousChoiceA, previousChoiceB;

    public Match(Agent agentA, Agent agentB){

        /* TODO :
            set this.agentA and this.agentB
            initialize previousChoices with UNDEFINED
        */

    }

    public void playGame() {

        // TODO : To make choice, each agent needs previous choice of the opponent
        ...
        // TODO : Update previous choices after the game

    }
}
```

src/Main.java

```
...  @@ -1,13 +1,19 @@
1    import evolution.of.truth.agent.Agent;
2    import evolution.of.truth.Match;
3    import evolution.of.truth.agent.Angel;
4  + import evolution.of.truth.agent.Copycat;
5    import evolution.of.truth.agent.Devil;
6
7    public class Main {
8        public static void main(String args[]) {
   -        Agent agentA = new Angel();
9  +        Agent agentA = new Copycat();
10           Agent agentB = new Devil();
   -        Match.playGame(agentA, agentB);
11 +        Match match = new Match(agentA, agentB);
12 +        match.playGame();
13 +        match.playGame();
14 +        match.playGame();
15 +        match.playGame();
16 +        match.playGame();
17           System.out.println(agentA.getScore());
18           System.out.println(agentB.getScore());
19       }
```

- Delete - lines from the code and add + lines

```
Main
"C:\Program Files\JetBrains\Intell
-1
3

Process finished with exit code 0
```

- A copycat won't be fooled by a devil for more than once!

Subprob 2-3 : Describing an agent (5 min)

- It will be helpful to override toString() of Object class

- Maybe we can override it for every `Agent` subclasses?
  - Too redundant :(

```
public class Angel extends Agent {
    @Override
    public String toString() {
        return "Angel: " + getScore();
    }
```

```
public class Devil extends Agent {
    @Override
    public String toString() {
        return "Devil: " + getScore();
    }
```

```
public class Copycat extends Agent {
    @Override
    public String toString() {
        return "Copycat: " + getScore();
    }
```

Subprob 2-3 : Describing an agent (5 min)

- Add String attribute `name` to the Agent
- Initialize `name` in the Agent constructor
  - For every Agent subclasses, call the parent class constructor with `super(name)`

- Override toString() function and return a string which contains the agent's `name` and `score`

src/Main.java

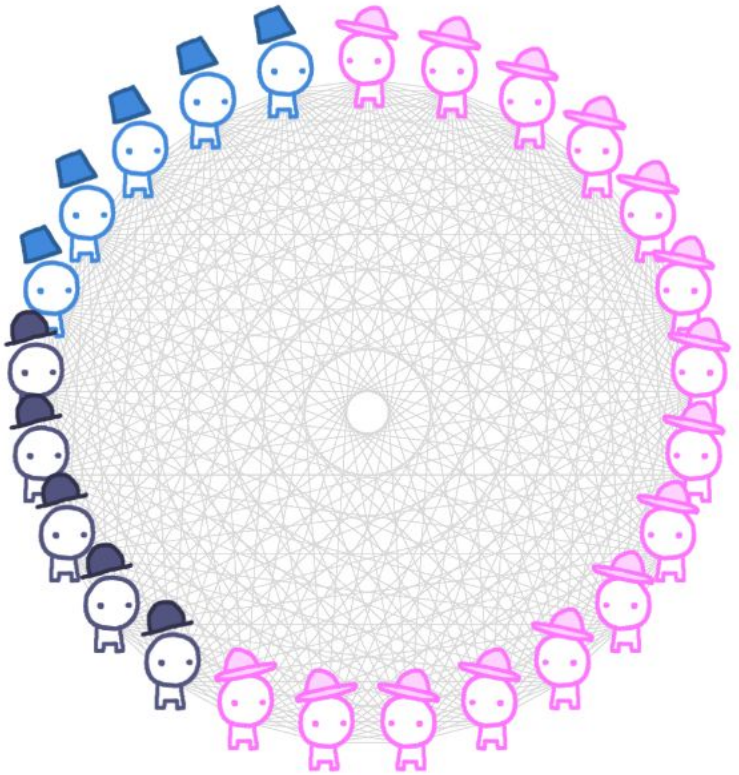
```
@@ -14,7 +14,7 @@ public static void main(String args[]) {
14          match.playGame();
15          match.playGame();
16          match.playGame();
-           System.out.println(agentA.getScore());
-           System.out.println(agentB.getScore());
17  +       System.out.println(agentA.toString());
18  +       System.out.println(agentB.toString());
19      }
20  }
```

```
Main
"C:\Program Files\JetBrains\IntelliJ
Copycat: -1
Devil: 3

Process finished with exit code 0
```

- Delete - lines from the code and add + lines

Subprob 3 : Pairwise match (5 min)

- We want to register a set of agents and play all games between them
- Also, we want to specify the number of games to play within each pair



Design Consideration: (1)

- The number of games within each pair
  - Option: class HundredMatches extends Match
    - Very Bad!
    - HundredAndOneMatches, HundredAndTwoMatches, ...
  - Instead, let's pass the number as a parameter of some function

Design Consideration: (2)

- Playing games for all pair of agents
  - Option 1: playGame(boolean allPairMatch)
    - Bad! Too different logic to be wrapped in a single function
  - Option 2: directly modify `Match` class
  - Option 3: class PairwiseMatch extends Match
  - Option 4: create class `Tournament` that uses `Match`

Option 2: directly modifying `Match`

- In fact, it is a good option for now.
- But we are in a practice session ;)
  - Let's assume the `Match` class is already being used in a number of different classes
  - Then we will want to keep it intact.

Option 3: inheriting from `Match`

- “Is every PairwiseMatch a Match?”
  - Not very clear
- A subclass must share all properties its parent has
  - However, `constructor(Agent, Agent)` does not make any sense to class `PairwiseMatch`

```
public class PairwiseMatch extends Match {
    public PairwiseMatch() {
        super( agentA: null,  agentB: null);
    }
}
```

Option 4: `Tournament` class

- Create class `Tournament` that uses `Match`
- `Tournament` class includes following methods
  - Match[] createAllMatches()
    - Create an array of all possible matches except for the match with same agent
    - So, there will be (`agents.length`)*(`agents.length`-1)/2 matches
  - void playAllGames(int numRounds)
    - At each round, play a game for all matches
    - Use `createAllMatches` method
  - void describe()
    - Call `.toString` method for all agents

Subprob 3 : Pairwise match (5 min)

evolution_of_truth.environment.Tournament.java

```
package evolution_of_truth.environment;

import evolution_of_truth.agent.Agent;
import evolution_of_truth.agent.Angel;
import evolution_of_truth.agent.Copycat;
import evolution_of_truth.agent.Devil;
import evolution_of_truth.match.Match;

public class Tournament {
    Agent[] agents;

    public Tournament() {
        agents = new Agent[25];
        for (int i = 0; i < 15; i++) {
            agents[i] = new Angel();
        }
        for (int i = 0; i < 5; i++) {
            agents[15 + i] = new Devil();
        }
        for (int i = 0; i < 5; i++) {
            agents[20 + i] = new Copycat();
        }
    }

...
```

```
...

   private Match[] createAllMatches() {
        ...
    }

    public void playAllGames(int numRounds) {
        ...
    }

    public void describe() {
        ...
    }
}
```

src/Main.java

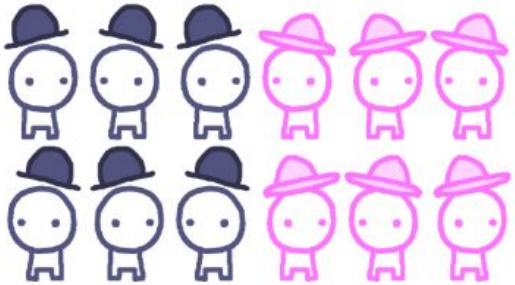
```
...  @@ -1,20 +1,13 @@
1     import evolution.of.truth.agent.Agent;
      - import evolution.of.truth.Match;
      - import evolution.of.truth.agent.Angel;
2   + import evolution.of.truth.environment.Tournament;
3   + import evolution.of.truth.match.Match;
4     import evolution.of.truth.agent.Copycat;
5     import evolution.of.truth.agent.Devil;
6
7     public class Main {
8         public static void main(String args[]) {
      -         Agent agentA = new Copycat();
      -         Agent agentB = new Devil();
      -         Match match = new Match(agentA, agentB);
      -         match.playGame();
      -         match.playGame();
      -         match.playGame();
      -         match.playGame();
      -         match.playGame();
      -         System.out.println(agentA.toString());
      -         System.out.println(agentB.toString());
9   +         Tournament tournament = new Tournament();
10  +         tournament.playAllGames(10);
11  +         tournament.describe();
12        }
13    }
```

```
Main
"C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.3\jbr\bin\java.exe"
Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 33
Process finished with exit code 0
```

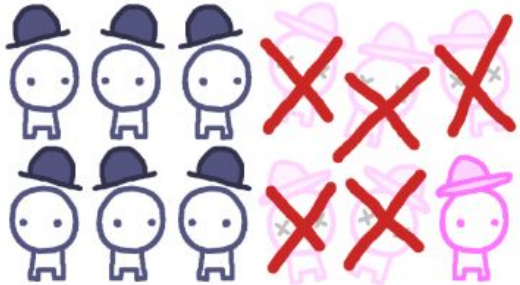
- Delete - lines from the code and add + lines

Subprob 4 : Evolving the population (10 min)

Now, let's let our population of players *evolve over time*. It's a 3-step dance:



1. PLAY A TOURNAMENT
Let them all play against each other, and tally up their scores.



2. ELIMINATE LOSERS
Get rid of the 5 worst players. (if there's a tie, pick randomly between them)

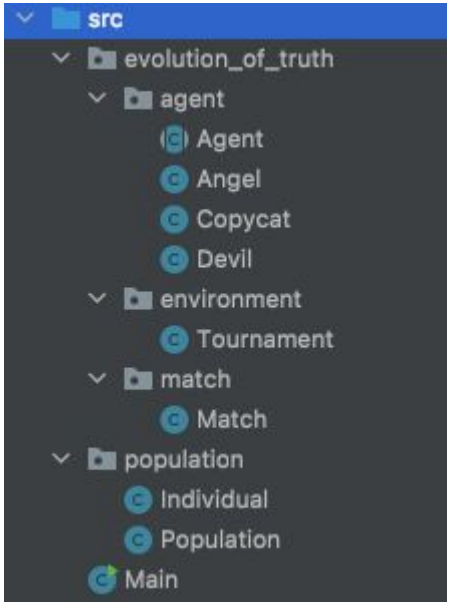
3. REPRODUCE WINNERS
Clone the 5 best players. (if there's a tie, pick randomly between them)

Subprob 4 : Evolving the population (10 min)

- This involves a bit cumbersome logic like
  - Sorting an array
  - Choosing top k and bottom k elements
  - Removing multiple elements in an array
- Luckily, you've found an external library that does the similar job
  - We can utilize this to reduce our work

Subprob 4 : Evolving the population (10 min)

- To simulate this, we wrote two classes for you (Special thanks to *Kiroong Choe*)
- Make a new package named `population`
  - create two classes `Population` and `Individual`
- Copy and paste the following code
  - Population (<- click this)
  - Individual (<- click this)



abstract class `Individual`

- protected Individual()
  - constructor
- abstract public int sortKey()
  - An integer to be used for sorting
- abstract public Individual clone()
  - It should return a copy of the object

class `Population`

- public Population()
  - Constructor
- public int size()
  - a size of the population
- Public Individual[] getIndividual()
  - returns an array of Individuals, sorted in ascending order
- public void addIndividual(Individual newIndividual)
  - Add a new Individual to the population
- public void toNextGeneration(int numReplace)
  - 1. Sort the population by sortKey()
  - 2. remove the lowest `numReplace` individuals
  - 3. clone the highest `numReplace` individuals

Integrating with our code

- Currently we have…
  - external package
    - class Population
    - class Individual (should be inherited)
  - our package
    - class Agent
    - class Tournament
- How should we compose them together?

Integrating with our code

- Think about relationship
  - A tournament **has** a population
  - Every agent **is** an individual
- So
  - Change `Agent[] agents` into `Population agentPopulation`
  - Let `Agent` be inherited from `Individual`
    - Implement sortKey() to return its score
    - Implement clone() to return a new instance of agent

src/evolution/of/truth/agent/Agent.java

```
...  @@ -1,6 +1,8 @@
1    package evolution.of.truth.agent;
2
     - abstract public class Agent {
3    + import kiroong.Individual;
4    +
5    + abstract public class Agent extends Individual {
6        private int score;
7        private String name;
8
M    @@ -9,6 +11,10 @@ protected Agent(String name) {
11           this.name = name;
12       }
13
14   +   public int sortKey() {
15   +       return getScore();
16   +   }
17   +
18       @Override
19       public String toString() {
20           return name + ": " + getScore();
```

population

```
public class Angel extends Agent {

    public Angel() {
        super("Angel");
    }

    @Override
    public Individual clone() {
        return new Angel();
    }
```

```
@Override
public Individual clone() {
    return new Copycat();
}
```

```
@Override
public Individual clone() {
    return new Devil();
}
```

src/evolution/of/truth/environment/Tournament.java

```
@@ -5,30 +5,45 @@
 import evolution.of.truth.agent.Copycat;
 import evolution.of.truth.agent.Devil;
 import evolution.of.truth.match.Match;
+import kiroong.Individual;
+import kiroong.Population;
 
 public class Tournament {
-    Agent[] agents;
+    Population agentPopulation;
 
     public Tournament() {
-        agents = new Agent[25];
+        agentPopulation = new Population();
         for (int i = 0; i < 15; i++) {
-            agents[i] = new Angel();
+            agentPopulation.addIndividual(new Angel());
         }
         for (int i = 0; i < 5; i++) {
-            agents[15 + i] = new Devil();
+            agentPopulation.addIndividual(new Devil());
         }
         for (int i = 0; i < 5; i++) {
-            agents[20 + i] = new Copycat();
+            agentPopulation.addIndividual(new Copycat());
         }
     }
```

population

```
    private Match[] createAllMatches() {
-        int n = agents.length;
+        int n = agentPopulation.size();
+        Individual[] agents = agentPopulation.getIndividuals();
         Match[] matches = new Match[n * (n - 1) / 2];
         int index = 0;
         for (int i = 0; i < n; i++) {
             for (int j = i + 1; j < n; j++) {
-                matches[index++] = new Match(agents[i], agents[j]);
+                matches[index++] = new Match((Agent)agents[i], (Agent)agents[j]);
             }
         }
         return matches;
@@ -44,8 +59,11 @@ public void playAllGames(int numRounds) {
     }
 
     public void describe() {
-        for(Agent agent: agents) {
+        Individual[] agents = agentPopulation.getIndividuals();
+        for(Individual _agent: agents) {
+            Agent agent = (Agent)_agent;
             System.out.print(agent.toString() + " / ");
         }
+        System.out.println();
     }
 }
```

src/evolution/of/truth/environment/Tournament.java

```
+     public void evolvePopulation() {
+         agentPopulation.toNextGeneration(5);
+     }
+
+     public void resetAgents() {
+         Individual[] agents = agentPopulation.getIndividuals();
+         for(Individual _agent: agents) {
+             Agent agent = (Agent)_agent;
+             agent.setScore(0);
          }
      }
```

src/Main.java

```
@@ -7,7 +7,11 @@
  public class Main {
      public static void main(String args[]) {
          Tournament tournament = new Tournament();
-         tournament.playAllGames(10);
-         tournament.describe();
+         for(int i=0;i<10;i++) {
+             tournament.resetAgents();
+             tournament.playAllGames(10);
+             tournament.describe();
+             tournament.evolvePopulation();
+         }
      }
  }
```

Copycat inherits the world!

Main

```
"C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.3\jbr\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ
Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel
Angel: 180 / Angel: 180 / Angel: 180 / Angel: 180 / Angel: 180 / Angel: 180 / Angel: 180 / Angel: 180 / Angel: 180 / Angel: 180 / Copyc
Angel: 30 / Angel: 30 / Angel: 30 / Angel: 30 / Angel: 30 / Devil: 165 / Devil: 165 / Devil: 165 / Devil: 165 / Devil: 165 / Copycat: 1
Devil: 15 / Devil: 15 / Devil: 15 / Devil: 15 / Devil: 15 / Devil: 15 / Devil: 15 / Devil: 15 / Devil: 15 / Devil: 15 / Devil: 15 / Dev
Devil: 30 / Devil: 30 / Devil: 30 / Devil: 30 / Devil: 30 / Devil: 30 / Devil: 30 / Devil: 30 / Devil: 30 / Devil: 30 / Devil: 30 / Dev
Devil: 45 / Devil: 45 / Devil: 45 / Devil: 45 / Devil: 45 / Devil: 45 / Devil: 45 / Devil: 45 / Devil: 45 / Devil: 45 / Copycat: 270 /
Devil: 60 / Devil: 60 / Devil: 60 / Devil: 60 / Devil: 60 / Copycat: 375 / Copycat: 375 / Copycat: 375 / Copycat: 375 / Copycat: 375 /
Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 /
Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 / Copycat: 480 /
```

Exercise

- Create a new agent `Copykitten`

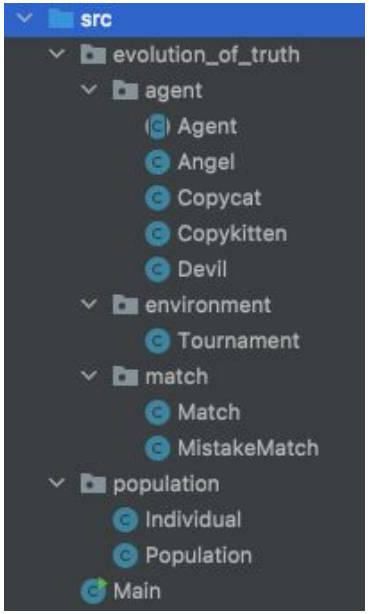
COPYKITTEN:
Hello! I'm like Copycat, except I Cheat back only after you Cheat me twice in a row. After all, the first one could be a mistake! Purrrrr

- Create a new class `MistakeMatch` inherited from `Match`
  - In this match, every agent's choice is reversed by 5% chance

- Replace all angels to copykittens in the population, and see if kittens prosper in the world with mistakes.

Submission

- Compress your `src` directory into a zip file.
  - After unzip, the 'src' directory must appear.
- Rename your zip file as 20XX-XXXXX_{name}.zip - for example, 2021-12345_JeongMinkyung.zip
- Upload it to eTL - Lab 5 assignment.



Directory Structure