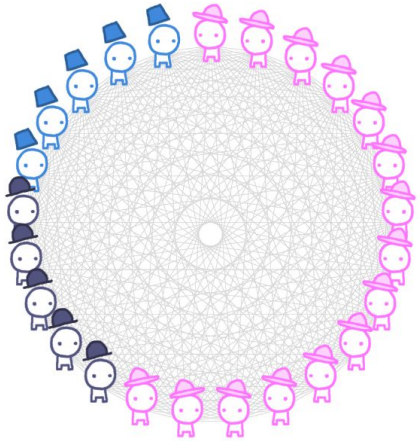
Inheritance

Lab 5

TA : Changmin Jeon, Minji Kim, Hyunwoo Jung,
Hyunseok Oh, Jingyu Lee, Seungwoo Jo

Topic: Evolution of Truth

- 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

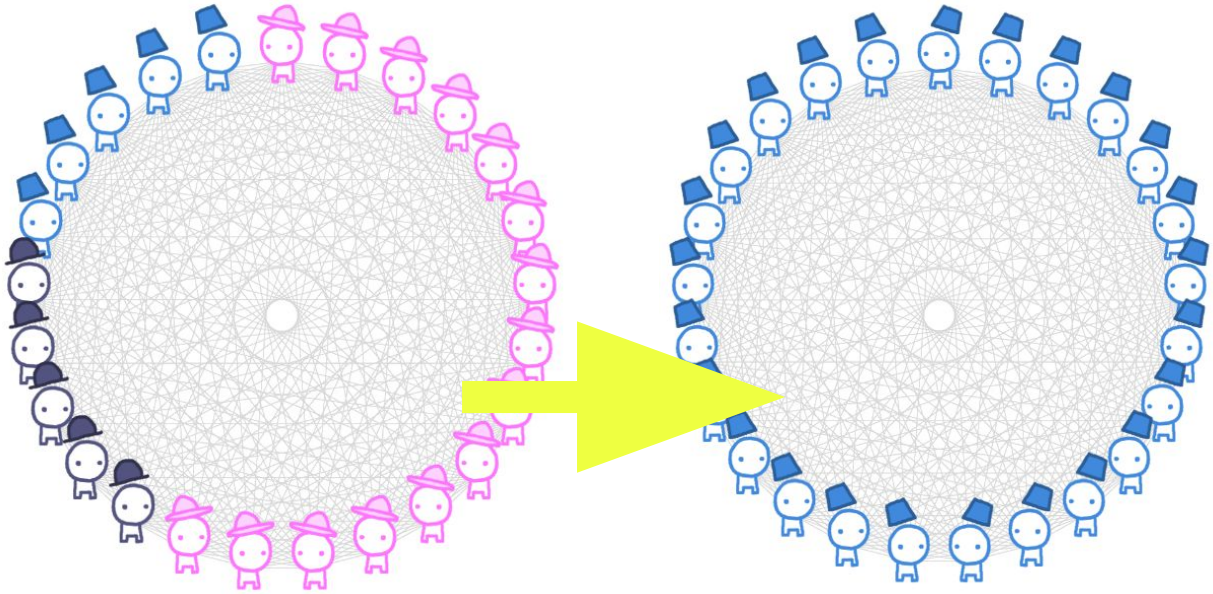
All Cheat

Copycat

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

Today's Goal

- 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

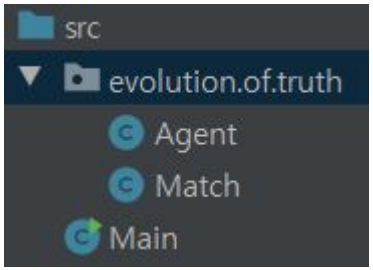
(oh, and by the way...)

Source codes

- You can start from scratch by following the steps in the slides!
- Reference
  - Entire project
  - Special thanks to *Kiroong Choe*

The simplest game

- Now let's make the simplest game
  - Two agents only choose to cooperate
- Project initialization
  - Make a package named `evolution.of.truth`
    - (Actually, it isn't good as a package name)
    - Inside, make two classes `Agent` and `Match`
  - Make a `Main` class at the top `src` directory
    - It is our entry point



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

```
@@ -0,0 +1,21 @@
+ package evolution.of.truth;
+
+ public class Agent {
+     private int score;
+
+     public Agent() {
+         score = 0;
+     }
+
+     public int getScore() {
+         return score;
+     }
+
+     public void setScore(int newScore) {
+         score = newScore;
+     }
+
+     public int choice() {
+         return Match.COOPERATE;
+     }
+ }
```

src/evolution/of/truth/Match.java

```
@@ -0,0 +1,24 @@
+ package evolution.of.truth;
+
+ public class Match {
+     public static int CHEAT = 0;
+     public static int COOPERATE = 1;
+
+     private static int ruleMatrix[][][] = {
+             {
+                     {0, 0}, // A cheats, B cheats
+                     {3, -1} // A cheats, B cooperates
+             },
+             {
+                     {-1, 3}, // A cooperates, B cheats
+                     {2, 2} // A cooperates, B cooperates
+             }
+     };
+
+     public static void playGame(Agent agentA, Agent agentB) {
+         int choiceA = agentA.choice();
+         int choiceB = agentB.choice();
+         agentA.setScore(agentA.getScore() + ruleMatrix[choiceA][choiceB][0]);
+         agentB.setScore(agentB.getScore() + ruleMatrix[choiceA][choiceB][1]);
+     }
+ }
```

src/Main.java

```
@@ -0,0 +1,12 @@
+ import evolution.of.truth.Agent;
+ import evolution.of.truth.Match;
+
+ public class Main {
+     public static void main(String args[]) {
+         Agent agentA = new Agent();
+         Agent agentB = new Agent();
+         Match.playGame(agentA, agentB);
+         System.out.println(agentA.getScore());
+         System.out.println(agentB.getScore());
+     }
+ }
```

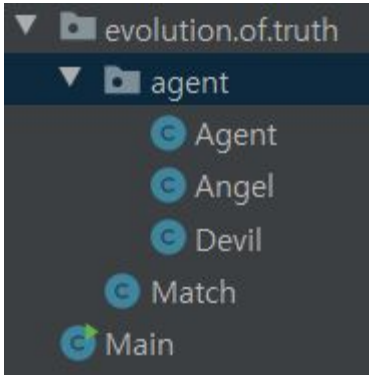
Main

```
"C:\Program Files\JetBrains\IntelliJ ID
2
2

Process finished with exit code 0
```

Angel vs. Devil


and


- 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`



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

```
...  @@ -0,0 +1,10 @@
1  + package evolution.of.truth.agent;
2  +
3  + import evolution.of.truth.Match;
4  +
5  + public class Angel extends Agent {
6  +     @Override
7  +     public int choice() {
8  +         return Match.COOPERATE;
9  +     }
10 + }
```

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

```
...  @@ -0,0 +1,10 @@
1  + package evolution.of.truth.agent;
2  +
3  + import evolution.of.truth.Match;
4  +
5  + public class Devil extends Agent {
6  +     @Override
7  +     public int choice() {
8  +         return Match.CHEAT;
9  +     }
10 + }
```

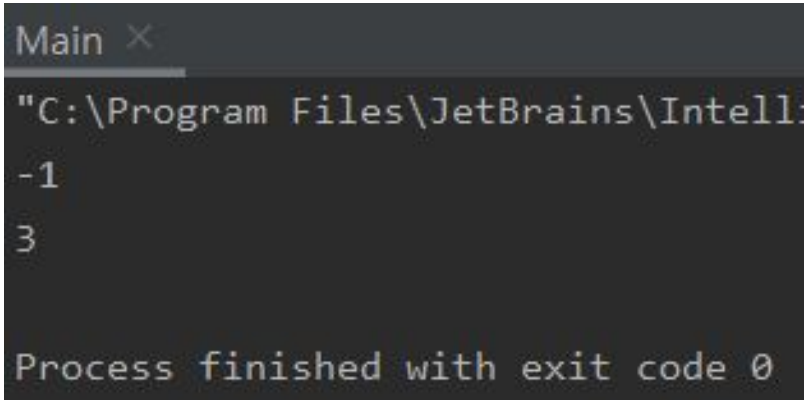
8 src/evolution/of/truth/Agent.java → src/evolution/of/truth/agent/Agent.java

```
@@ -1,6 +1,6 @@
- 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() {
          return score;
      }

      public void setScore(int newScore) {
          score = newScore;
      }

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

- 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.

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());
```



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

Copycat

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

```
public class Copycat extends Agent {
    @Override
    public int choice() {
        // ?????
    }
}
```

- Copycat needs a previous choice of an opponent
- But our `choice` function doesn't have any parameters

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

```
@@ -15,5 +15,5 @@ public void setScore(int newScore) {
15          score = newScore;
16      }
17
-     abstract public int choice();
18  +   abstract public int choice(int previousOpponentChoice);
19  }
```

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

```
@@ -4,7 +4,7 @@
4
5   public class Angel extends Agent {
6       @Override
-       public int choice() {
7   +       public int choice(int previousOpponentChoice) {
8           return Match.COOPERATE;
9       }
10  }
```

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

```
@@ -4,7 +4,7 @@
4
5   public class Devil extends Agent {
6       @Override
-       public int choice() {
7   +       public int choice(int previousOpponentChoice) {
8           return Match.CHEAT;
9       }
10  }
```

src/evolution/of/truth/Match.java

```
@@ -5,6 +5,7 @@
5   public class Match {
6       public static int CHEAT = 0;
7       public static int COOPERATE = 1;
8   +   public static int UNDEFINED = -1;
9
```

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

```
...  @@ -0,0 +1,14 @@
1   + package evolution.of.truth.agent;
2   +
3   + import evolution.of.truth.Match;
4   +
5   + public class Copycat extends Agent {
6   +     @Override
7   +     public int choice(int previousOpponentChoice) {
8   +         if (previousOpponentChoice == Match.UNDEFINED) {
9   +             return Match.COOPERATE;
10  +         } else {
11  +             return previousOpponentChoice;
12  +         }
13  +     }
14  + }
```

Copycat

- 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!

```
21 +     Agent agentA, agentB;
22 +     int previousChoiceA, previousChoiceB;
23 +
24 +     public Match(Agent agentA, Agent agentB) {
25 +         this.agentA = agentA;
26 +         this.agentB = agentB;
27 +         previousChoiceA = UNDEFINED;
28 +         previousChoiceB = UNDEFINED;
29 +     }
30 +
31 +     public void playGame() {
32 +         int choiceA = agentA.choice(previousChoiceB);
33 +         int choiceB = agentB.choice(previousChoiceA);
34           agentA.setScore(agentA.getScore() + ruleMatrix[choiceA][choiceB][0]);
35           agentB.setScore(agentB.getScore() + ruleMatrix[choiceA][choiceB][1]);
36 +         previousChoiceA = choiceA;
37 +         previousChoiceB = choiceB;
38       }
39   }
```

- From now on, we will make an instance of Match for every pair of agents

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       }
```

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!

Describing an agent

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

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

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

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

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

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

```
@@ -2,9 +2,16 @@

 abstract public class Agent {
     private int score;
+    private String name;

-    public Agent() {
+    protected Agent(String name) {
         score = 0;
+        this.name = name;
     }
+
+    @Override
+    public String toString() {
+        return name + ": " + getScore();
     }

     public int getScore() {
```

```
public class Angel extends Agent {
+    public Angel() {
+        super("Angel");
+    }
```

```
public class Devil extends Agent {
+    public Devil() {
+        super("Devil");
+    }
```

```
public class Copycat extends Agent {
+    public Copycat() {
+        super("Copycat");
+    }
+
```

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  }
```

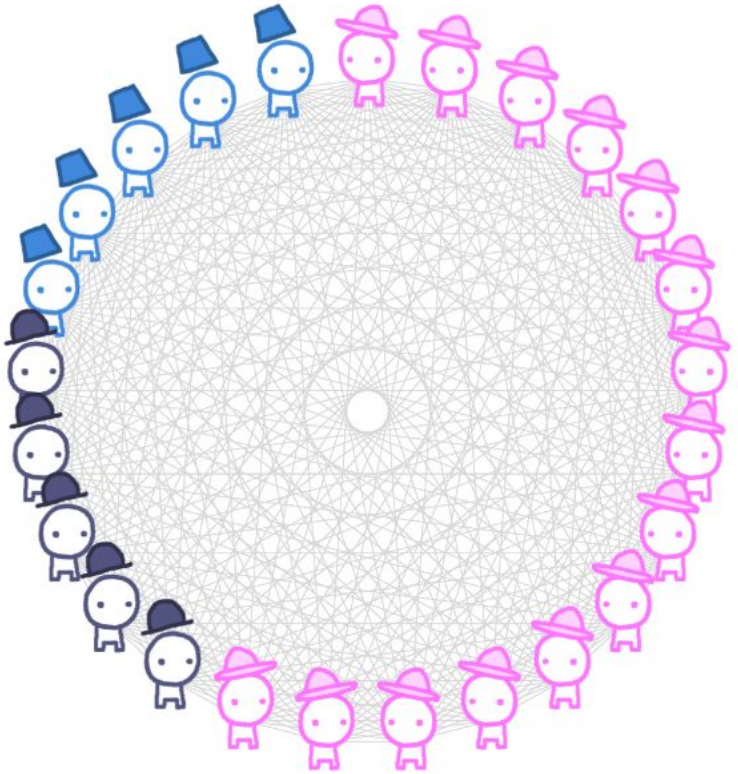
Main

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

Process finished with exit code 0
```

Pairwise match

- 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 paramter 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.

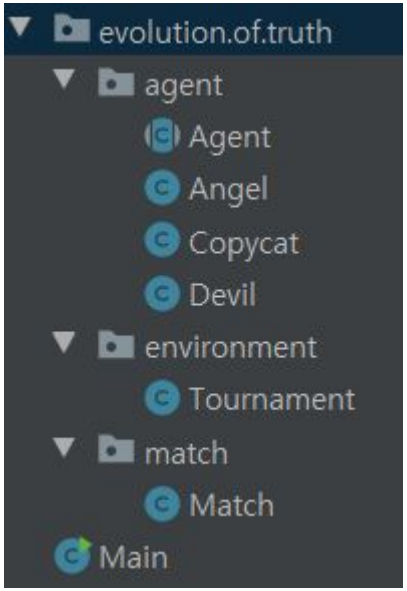
Option 3: inheriting from `Match`

- “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

- class Tournament
  - ~~void registerAgents(Agent[] agents)~~
    - Try it yourself, later!
  - void playAllGames(int numRounds)
  - void describe()



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

```
@@ -0,0 +1,51 @@
+ 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() {
+         int n = agents.length;
+         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]);
+             }
+         }
+         return matches;
+     }
+
+     public void playAllGames(int numRounds) {
+         Match[] matches = createAllMatches();
+         for (int round = 0; round < numRounds; round++) {
+             for (Match match : matches) {
+                 match.playGame();
+             }
+         }
+     }
+
+     public void describe() {
+         for(Agent agent: agents) {
+             System.out.print(agent.toString() + " / ");
+         }
+     }
+ }
```

src/Main.java

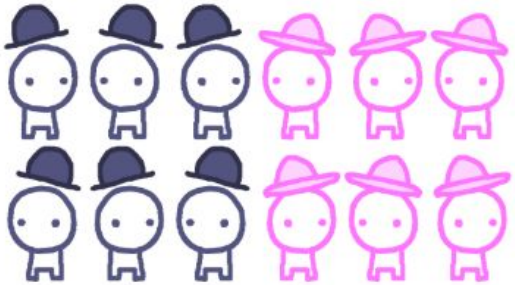
```
@@ -1,20 +1,13 @@
  import evolution.of.truth.agent.Agent;
- import evolution.of.truth.Match;
- import evolution.of.truth.agent.Angel;
+ import evolution.of.truth.environment.Tournament;
+ import evolution.of.truth.match.Match;
  import evolution.of.truth.agent.Copycat;
  import evolution.of.truth.agent.Devil;

  public class Main {
      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());
+         Tournament tournament = new Tournament();
+         tournament.playAllGames(10);
+         tournament.describe();
      }
  }
```

Main

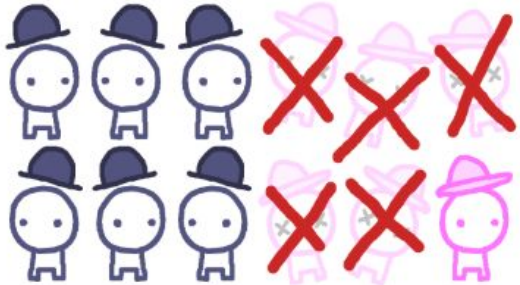
```
"C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.3\jbr\bin\java.exe"
Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 330 / Angel: 33
Process finished with exit code 0
```

Evolving the population

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)

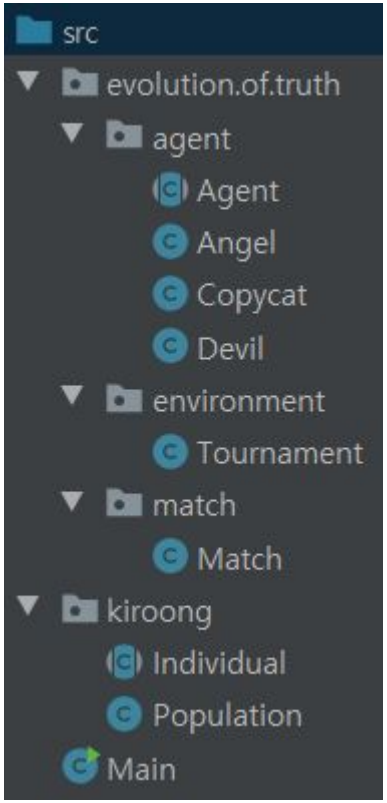
3. REPRODUCE WINNERS
Clone the 5 best players. (if there's a tie, pick randomly between them)

Evolving the population

- 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

Evolving the population

- To simulate this, I wrote two classes for you
- Make a new package named `kiroong`
  - 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();
```

```
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());
         }
     }
```

```
     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, 2020-12345_KimMinji.zip
- Upload it to eTL - Lab 5 assignment.

Directory Structure