Don’t Fear OOP

Noam Silverman and Erfan Yeganehfar

In Java OOP stands for Object Oriented Programming. For beginners OOP may be a hard concept to understand, so to make things easier to understand we can use a government as a way of representing how OOP can work in Java. Through a story about a government (The Sad Government: A Novel) involving a head of state, bribes, and angry twitter messages. Through this story we will learn how to understand OOP in a simple way.

To avoid having to read things that are too complicated or simple for you, green writing is made for beginners who really know nothing about OOP, brown has its example’s written in almost English to allow for people to read more closely to how it is written in Java (pseudo code), but without requiring them to understand Java code, and red is the hardest to read, because the examples are written in actual Java code. All words in **bold** are real Java terms, so you may want to remember them.

This is a story of a Sad Government (A Novel) set in the country of Countria, yet for a Sad Government to exist there needs to be a default blueprint for any Government. That’s why a default Government is first “described” to set down the base “rules”.

|  |
| --- |
| Every Government has a large number of politicians, the ruling party, a Government type, a country, a head of state title, and is located in the capital. The typical government has 1 congress, and 1 head of state. |

As seen in the seen in the box, the components of a Government were set in a general format (which can later be changed). It also gave the default values for a typical Government, which has only 1 head of state and 1 congress. This is called a **class**, and by setting the empty **variables** you can make many new and different governments called **objects**.

|  |
| --- |
| Government  has a certain number of politicians  has a certain number of heads of state with their respective titles  has a certain ruling party  has a certain Government type  has a main location of governing  has a country  has a capital  a typical Government has  heads of state = 1  congress = 1 |

As expected a Government would be defined by certain amounts of politicians, head of state, a certain party etc. These are all types of variables that are integral to any Government. Beneath the section for a general government, we see that the values for a typical have been set. Default governments have only 1 head of state and 1 congress. This is called a **class**, and by setting the empty **variables** you can make many new and different governments called **objects**.

|  |
| --- |
| Public class Government  {  int politiciansNum;  String stateHeadTitle;  String rulingParty;  String govType;  int stateHead;  String location;  int congress;  String country;  String capital;  public WesternTown()  {  stateHead= 1;  congress = 1;  }  } |

To create a blueprint of a Government, the Government **class** is used. When talking about OOP, classes are used as “blueprints” for objects. They house their **methods** (actions) and their **instance variables** (attributes). In the Government class created above, there exists components needed for a Government without specifying what government it is. The bare elements (**instance variables**) are defined, yet not all of them are given specific values, allowing fluidity in the type of Government. (For now, Data types such as “int” and “String” store data “in” a specified **variable**) These variables give the default value for any Government object made inside the **constructor**, yet allows the other **instance variables** to be manipulated later on.

The **constructor** in the **class** determines the default values for some of the **instance** variables in the **class**, it also acts as a creator of new objects when an object is **instantiated** as a Government object.

**“;”** specifies the end of **statements** and **“{}”** specifies a block of code where statements can be in relevance to the location.

If a Government object is created through **instantiation** (creating a new instance of said object), then all the **attributes** of a Government would be based off of the Government **class**.

So after the “rules” for a Government are made, it's time to actually create a specific type of Government using the Government **class**, in this case it is the Government of Countria represented by the SadGovernment (**class**) novel.

|  |
| --- |
| SadGovernment  Government Of Countria is a new Government  The government is a very large government of 469 politicians (by complete coincidence, that is the size of the US government's house and senate). The ruling party is called “The Party That Rules!” and it is the government of the great nation of Countria. The government type is a republican democracy headed by a President, and is based out of the wonderous city of Aelia Capitolina. |

Here it can be observed that indeed a new Government is made. Since the Government of Countria is a Government, it shares the same values with the Government **class**. It also has certain specified values that were not stated within the default Government allowing it to be unique and more specific as it gives more information about the Government.

|  |
| --- |
| Main SadGovernment  Government Of Countria is new Government  The number of politicians is 469  The ruling party is “The Party That Rules!”  The name of the country is Countria  The government type is republican democracy  The head of state’s title is “President”  The capital is “Aelia Capitolina” |

The Government of the story is a government called the Government Of Countria. This means that it “has” all the same variables as the Government **class**. However, if you remember from above, the Government **class** had **instance variables** (the **variables** that are features of a **class**) that were left open ended, like the number of politicians, the capital name, etc. Since we have now made a specific government, all those **variables** have been filled in.

|  |
| --- |
| public class SadGovernment  {  public static void main (String args[])  {  Government govOfCountria = new Government;  govOfCountria.politiciansNum = 469;  govOfCountria.rulingParty = “The Party That Rules!”;  govOfCountria.country = “Countria”;  govOfCountria.govType = “Republican Democracy”;  govOfCountria.stateHeadTitle = “President”;  govOfCountria.capital = “Aelia Capitolina”;  }  } |

The first thing to notice is the is Government govOfCountria = new Government. This means that a new Government **object**, called govOfCountria has been created (**instantiated)** . Pretty simple. This new government is an **object** of the Government **class**. How does that work, you ask? Imagine the **class** as a cookie cutter, and all the **objects** as cookies. The cookie cutter of Government was used to cut out Government shaped cookies. This particular cookie we have just cut out is called govOfCountria. But just because two cookies are the same in shape, does not mean they are the same in every other way. A star shaped cookie cutter can be used to cut out the shape for an oatmeal cookie, a chocolate cookie, or even an inedible cookie made out of clay. Same shape, but very different cookies.

The number of politicians, the ruling party, the country, and more were left to the makers discretion. The part beneath Government govOfCountria = new Government lets the user do that. The variable name on the other side of the period is the empty variable now being assigned. By the end of the above code, all empty variables have been set to make a well defined (and specific) object.

Now that the general Government is formed, some extra objects can be added to the book to make it a little more detailed. As politicians in general are a huge part of what makes a government the example shown below is of the Politicians **class**.

Here is the base definition of politicians in this book:

|  |
| --- |
| All politicians have two legs, two arms, eyes, a nose, a mouth, and no brain or heart. They are either male or female, have a name, and have a party affiliation. If someone asks for a their name, politicians can respond. |

This part here is really just the same as the first **class**, but instead of making the template for a Government, this makes the template for a Politician. It makes the values for a Politician, some such as party affiliation can be set by the maker of a specific politician, and some such as the number of eyes are things that standard Politicians have (or lack). There is also something else though. The last sentence says that Politicians perform actions. If someone asks a Politician“Hey, what is your name?” Politician would say “My name is [name]”.

|  |
| --- |
| Politicians  have a certain number of legs  have a certain number of arms  have a certain number of eyes  have a certain number of noses  have a certain number of mouths  have a name have a certain sex  have a name have a certain number of hearts  have a name have a certain number of brains  have a political party affiliation  A standard politician would start with  two legs  two arms  two eyes  one nose  one mouth  no hearts  no brains  When someone asks for your name  tell them your name |

Every government needs politicians. This box shows how the Politicians are made. They have a long list of **attributes** (in the form of **instance variables**) such as arms. Underneath we see that standard (default) politicians have default values for various **attributes** such as the number of legs. So far, the same as with the government.

However, the last section is very important. The previous sections have created Politicians to be like this or that, but now this part makes them actually able to do something (an action). If someone asks the politician their name, they respond with it.

|  |
| --- |
| public class Politicians  {  int legs;  int arms;  int eyes;  int nose;  int mouth;  String name;  String sex;  String politicialParty;  int brains;  int hearts;    public Humans()  {  legs = 2;  arms = 2;  eyes = 2;  nose = 1;  mouth = 1;  hearts = 0;  brains = 0;  }  public String whatIsYourName()  {  return name;  }  } |

This part makes Politicians, much the same way that it did with Governments. We have a Politician **class** with **instance variables**, and a **class** **constructor** where certain default **fields** are given a value**.**

However, if you notice the last part, it is something new. It is a **method.** All previous discussed topics have made a politician have something (describe them), like two eyes, now politicians can be made to do something (an action). This **method** lets politicians respond with their name if asked for it.

This description of Politicians although plenty, is much like Governments, not very specific. In this book many types of Politicians exist, one such being the Head of State.

|  |
| --- |
| Heads of states are extensions on the idea of politicians. They share the same default attributes, except that they have some additional qualities, specifically their favourite suit, some level of corruptness, a certain amount of bills signed, and a certain number of politicians they have attacked on twitter. Your standard head of state starts their term uncorrupt, have signed no bills and have not attacked any politicians on twitter. |

The first idea to grasp in the box is that Heads of states are Politicians. They **inherit** all the same variables as Politicians, meaning that they are just Politicians with new features tacked on. For example, they sign bills even though other Politicians don’t, while still having a name, a number of eyes, and so on.

The last sentence gives the default variables for a standard Head of State when they start their term, they have signed no bills.

|  |
| --- |
| Heads of state extends the idea Politicians  have a favorite suit  have a level of corruptness  have a certain amount of bills signed  have a certain number of politicians they have attacked on twitter  A standard head of state would start with  a corruption level of 0  no bills signed  no politicians they have attacked on twitter |

The main idea of this box is that Heads of States are just Politicians with extra details. They are at their core, Politicians with eyes, arms, legs, party affiliations, etc. However, the Head of State gets new variables such as number of bills signed, which regular Politicians do not have.

The last section gives the default variables for a standard Head of State when they start their term, they have signed no bills.

|  |
| --- |
| public class HeadsOfState extends Politician  {  String favSuit;  int corruptionLevel;  int billsSigned;  int politiciansAttackedOnTwitter;  Politicians attackedPolitician;  public HeadsOfState()  {  corruptionLevel = 0;  billsSigned = 0;  politiciansAttackedOnTwitter = 0;  }  } |

This is the general idea of **subclasses**. As “Politician” was a **class** and “HeadsOfState” being a **subclass** of it. Allowing the HeadOfState **class** to **inherit** the **attributes** and **behaviours** of the parent **class** Politician. Here it can also be seen that the **instance variable** “attackedPolitician” is declared as a Politician, this is done so that when the Head of State “attacks” a Politician on twitter, it is known that an attackedPolitician is a Politician.

Now that the Head of State is thoroughly explained it’s time to now move onto it’s **behaviours** (or **methods**). As Heads of State are the most powerful position in any Government, they sure do love to keep that power (even though it may be illegal). So a method can be made to show that a politician has done something illegal.

|  |
| --- |
| Whenever the head of state does something illegal, his level of corruptness will go up by one. |

These methods are different than just describing a Politician like before, as it shows how this certain Politician acts (**methods**,or also known as **behaviours**).

|  |
| --- |
| doIllegalAction  corruptness goes up by one |

Like variables, methods have an **identifier** and a “description” of what they are going to do. You may also notice how the header is a verb, this is because it is describing an action.

|  |
| --- |
| public void doIllegalAction()  {  corruptnessLevel++;  } |

When the Head of State does something illegal, the variable “corruptnessLevel” increases by one (signified by the “++” after the variable).

Heads of States are also well known for their masterful lies, so it isn't surprising to hear that when they are called out for being corrupt, they deny it. In this case a method can be used to **return** a message of how they aren't corrupt.

|  |
| --- |
| If someone asks a head of state how corrupt they are, the head of state will always say they are not corrupt even if it means lying. |

|  |
| --- |
| When someone asks for your corruption level  Quick! Tell them there was no collusion or something! |

The politician will always maintain that they are not corrupt, no matter how distant from the truth that may be.

|  |
| --- |
| public String howCorruptAreYou()  {  return “When the head of state does it, that means it’s not illegal”;  } |

If you thought this was another **method** for the Head of State, your correct! This is like the respond with your name **method** where the politician answers with their name when asked. The Head of State has **inherited** that **method** too of course, but this new **method** gets them to respond to questions about their foreign real estate or something. The **String** data type is in the method header to signify that this **method** will **return** a **String**. If asked they will always respond (**return**) with “When the head of state does it, that means it’s not illegal.”

The Head of State is starting to be a very fleshed out and detailed character within the story, yet because this is a Sad Government the Head of State should get involved in wrathful twitter messages, just to make the Government even sadder. Yet another method can be used to represent this, but in this case the method needs information about another Politician to bash. To make the book more flexible the Politician's name is a variable.

|  |
| --- |
| If the head of state is supposed to attack a politician on twitter, they unleashes tweet filled with grammatical and spelling errors, then adds one to the number of politicians they have attacked. Then print out "[Specified politician] is a bad hombre.'' |

Another **method**. This one lets the Head of State slander other Politicians. When they do so, the number of politicians they have attacked goes up by one.

|  |
| --- |
| attackPoliticianOnTwitter  add one to the number of politicians attacked  print "[Specified politician] is a bad hombre.'' |

This is another **method**. In this one, the Head of State attack other Politicians on twitter, and the number of politicians attacked increases by 1. (Pretty much the same thing as the green text)

|  |
| --- |
| public void attackPoliticianOnTwitter (Politician attackedPolitician)  {  this.attackedPolitician = attackedPolitician;  politiciansAttackedOnTwitter++;  System.out.println (attackedPolitician.whatIsYourName() + “ is a bad hombre.”);  } |

In this **method**, the Head of State attacks other Politicians on twitter, and the number of politicians attacked increases by 1. Pretty standard so far.

You may be wondering about the “Politician attackedPolitician” in between the brackets. That is called **argument**. If we imagine for a second that there was no argument, the mean tweet would be pretty useless, because what this **argument** does is it indicates the target of the twitter wrath. When ever this method occurs, it would go along with a Politician called attackedPolitician and everyone would know the victim. The **parameters** (**arguments**) and **instance variables** are the same. So, we are using the “**this.**” keyword to distinguish between **local variables,** (**variables** that can only be accessed inside 1 method)and **instance variables**.

To recap everything we have learned with the HeadOfState **class** (don’t worry, nothing new was added):

|  |
| --- |
| Heads of states are based on the idea of politicians. They share the same default attributes, except that they have some additional qualities, specifically their favourite suit, some level of corruptness, a certain amount of bills signed, and a certain number of politicians they have attacked on twitter. Your standard head of state starts their term uncorrupt, have signed no bills and have not attacked any politicians on twitter.  Whenever the head of state does something illegal, his level of corruptness will go up by one.  If someone asks a head of state how corrupt they are, the head of state will always say they are not corrupt, even if it means lying.  If the head of state is supposed to attack a politician on twitter, they unleashes tweet filled with grammatical and spelling errors, then adds one to the number of politicians they have attacked. Then print out "[Specified politician] is a bad hombre.'' |

|  |
| --- |
| Heads of state extends the idea Politicians  have a favorite suit  have a level of corruptness  have a certain amount of bills signed  have a certain number of politicians they have attacked on twitter  A standard head of state would start with  a corruption level of 0  no bills signed  no politicians they have attacked on twitter  Illegal Action  corruptness goes up by one  When someone asks for your corruption level  Quick! Tell them there was no collusion or something!  attackPoliticianOnTwitter  add one to the number of politicians attacked  print "[Specified politician] is a bad hombre.'' |

|  |
| --- |
| public class HeadsOfState extends Politician  {  String favSuit;  int corruptionLevel;  int billsSigned;  int politiciansAttackedOnTwitter;    public HeadsOfState()  {  corruptionLevel = 0;  billsSigned = 0;  politiciansAttackedOnTwitter = 0;  }  public void doIllegalAction()  {  corruptnessLevel++;  }  public String howCorruptAreYou()  {  return “When the head of state does it, that means it’s not illegal”;  }  public void attackPoliticianOnTwitter (Politician attackedPolitician)  {  this.attackedPolitician = attackedPolitician;  politiciansAttackedOnTwitter++;  System.out.println (attackedPolitician.whatIsYourName() + “ is a bad hombre.”);  }  } |

Hopefully that all makes sense, since we covered each topic individually, and are now just putting them together in one box. If anything is confusing, go back to the box of the topic that confuses you and reread it. It may take a second look to understand.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

At the beginning, we talked about how this was a novel called The Sad Government. We have created the character templates, and the government that this story happens in. Now lets see how this all plays out. This part is called the **main routine.**

|  |
| --- |
| Here is the main Sad Government that’s been constructed. There is a Government called the Government Of Countria. The Countria Government has a very large government of 469 politicians, the ruling party is called “The Party That Rules!” and it is the government of the great nation of Countria. The government type is a republican democracy headed by a President, and is based out of the wonderous city of Aelia Capitolina.  There is a Head of State named Erfan Silverman. He is a Politician from The Party That Rules! He has two legs, two arms, eyes, a nose, a mouth, and no brain or heart. His favourite suit is a Navy Blue suit from Silver Suits, and at the start of his term he is uncorrupt and has signed no bills into laws. He has also not yet attacked any other politicians on twitter.  One day, President Erfan Silverman takes a shady bribe, increasing his corruptness level. After being called out for his illegal action he denies all possibilities of it saying “When the head of state does it, that means it’s not illegal”. Later on he angrily tweets about Mop Yeganehfar, redirecting blame and calling him out, writing “Mop Yeganehfar is a bad hombre.”  THE END |

|  |
| --- |
| Here is the main plot in the story The Sad Government;  The Government is the Government of Coutria  The Government of Coutria has 469 politicians  The Government of Coutria’s ruling party is “The Party That Rules!”  The name of the country The Government of Coutria rules is Countria  The Government of Coutria is a republican democracy  The head of state’s title is “President”  The capital of The Government of Coutria is “Aelia Capitolina”  The Head of State is named Erfan Silverman  Erfan Silverman’s favourite suit is a Navy Blue Suit he got from Silver Suits  Erfan Silverman has signed no bills into law  Erfan Silverman is male  Erfan Silverman is from The Party That Rules!  There is a new Politician named Mop Yeganehfar  Mop Yeganehfar is male  Mop Yeganehfar is from The Party Of Democracy |

|  |
| --- |
| public class SadGovernment  {  public static void main (String args[])  {  Government govOfCountria = new Government;  govOfCountria.politiciansNum = 469;  govOfCountria.rulingParty = “The Party That Rules!”;  govOfCountria.country = “Countria”;  govOfCountria.govType = “Republican Democracy”;  govOfCountria.stateHeadTitle = “President”;  govOfCountria.capital = “Aelia Capitolina”;  HeadOfState erfanSilverman = new HeadOfState();  erfanSilverman.favSuit = “Navy Blue suit from Silver Suits”;  erfanSilverman.name = “Erfan Silverman”;  erfanSilverman.sex = “Male”;  erfanSilverman.politicialParty = “The Party That Rules!”;  Politician mopYeganehfar = new Politician();  mopYeganehfar.name = “Mop Yeganehfar”;  mopYeganehfar.sex = “Male”;  mopYeganehfar.politicialParty = “The Party Of Democracy”;  erfanSilverman.doIllegalAction();  System.out.println(erfanSilverman.howCorruptAreYou());  erfanSilverman.attackPoliticianOnTwitter(Politician mopYeganehfar);  }  } |

OOP has advantages to writing a story normally. Let’s say you want people to have a different name, like Erfan Silverman to become Efran Silverman. In normal writing, you have to go through the story and find every use of that name and correct it. However, with the power of OOP, you can simply change the variable String name for the object, and bam, everywhere the name is used, it changes! Much easier. This can be applied for any other situation you want to change things as well. Very nice!