THE LIVING GRIMOIRE

BY MOTI BARSKI

about the programmer that created the living grimoire : Moti Barski, Battle programmer.

name of the software : living grimoire : LG for short
what it is : an A.G.I platform.

intro :

how could coders have been coding this way for so long ?
they get some lame project, and they gotta start all over again building from scratch :

menues, the basics, rethink the algorithms and how to fit them into the small and big picture.
no matter how many projects you finished, with each project you would have to start over.

over time you would remember the main thinking patterns for solving puzzles BUT !,
codes and mini algorithms, there is no way to remember all of that.
you try to keep up to date with the latest codes and walkthroughs to no avail as
they expire you forget them and have to search for them again.
with doing the above you waste so much time that by the time you finish, the codes
you learnt are obsolete, in other words you are chasing rainbows.

like a carpet being pulled from under you, you gotta now readdapt your algs and codes
all over again to the new project.

even with design patterns many coders find that they need to adapt to them rather the other
way.

here it is different, it is truely amazing !

with this new way, battle programming, you are in a 7 star hotel in the buffet
and all you have to do is pick the skills you want and need for your project.
then assemble said skills with just one line of code per skill.
next, you can enjoy a nice anime or bike ride, cause your project is done.

****
****

LINKS :

ALL CLASSES :

https://github.com/yotamarker/public-livingrimoire

FORUM FOR LIVING GRIMOIRE SKILLS :

http://livingrimoire.com/forum/

PRINCIPLE OF WORK :

https://www.youtube.com/watch?v=0bxmsuxxcxu

EXPLAINED BY KEEN :

https://streamable.com/pndv8

EXPLAINED BY KERMIT

https://streamable.com/sh7ag

EXPLAINED BY MAD SCIENTIST :

https://streamable.com/7pz4j

method of operation overview: the LG can absorb skills and use them

what is a skill ?

a skill cosists of 2 factors : a summoner and actions.

summoner (referred to as a Dclass (a class whos name starts with D)):
input passing through a DClass can trigger the summoning of an algorithm
with actual time OR ear, eye, and other types of input as a string.

what is an algorithm ? a combination of alg parts

what is an alg part ?
an alg part is an action. an alg part class name starts with AP.
at each think cycle said action does one thing.

example DSayer class :

Code:

```
import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import PKGsystemClasses.APSay;
import PKGsystemClasses.AbsAlgPart;
import PKGsystemClasses.Algorithm;
import PKGsystemClasses.Neuron;
import PKGsystemClasses.Neuronable;

public class DSayer extends AbsCmdReq implements Neuronable {
 private int times;
 private String param;

 public DSayer() {
 super();
 this.times = 1;
 this.param = "";
 }

 public static String regexChecker(String theRegex, String str2Check) {
 Pattern checkRegex = Pattern.compile(theRegex);
 Matcher regexMatcher = checkRegex.matcher(str2Check);
 while (regexMatcher.find()) {
 if (regexMatcher.group().length() != 0) {
 return regexMatcher.group().trim();
 }
 }
 return "";
 }

 @Override
 public void input(String command) {
 int foo = 1;
 String myString = regexChecker("(\\d+)(?= times)", command);
 String toSay = regexChecker("(?<=say)(.*)(?=\\d)", command);
 if (myString != "") {
 foo = Integer.parseInt(myString);
 } else {
 toSay = regexChecker("(?<=say)(.*)", command);
 }
 this.param = toSay;
 this.times = foo;
 }

 @Override
 public void output(Neuron noiron) {
 // TODO Auto-generated method stub
 if (!param.isEmpty()) {
 AbsAlgPart itte = new APSay(this.times, this.param);
 String representation = "say " + param;
 if (this.times > 1) {
```

```
 representation += " " + this.times + " times";
 }
 ArrayList<AbsAlgPart> algParts1 = new ArrayList<>();
 algParts1.add(itte);
 Algorithm algorithm = new Algorithm("say", representation, algParts1);
 noiron.algParts.add(algorithm);
 }

 }
}
```

example APclass :

Code:

```
/* it speaks something x times
 * a most basic skill.
 * also fun to make the chobit say what you want
 * */
public class APSay extends AbsAlgPart {
    protected String param;
    private int at;

    public APSay(int at, String param) {
        super();
        if (at > 10) {
            at = 10;
        }
        this.at = at;
        this.param = param;
    }

    @Override
    public String action(String input) {
        // TODO Auto-generated method stub
        String axnStr = "";
        if (this.at > 0) {
            if (!input.equalsIgnoreCase(param)) {
                axnStr = param;
                at--;
            }
        }
        return axnStr;
    }

    @Override
    public enumFail failure(String input) {
        // TODO Auto-generated method stub
        return enumFail.ok;
    }

    @Override
    public Boolean completed() {
        // TODO Auto-generated method stub
        return at < 1;
    }

    @Override
    public AbsAlgPart clone() {
        // TODO Auto-generated method stub
        return new APSay(this.at, this.param);
    }

@Override
public Boolean itemize() {
// TODO add logic
// at home
return true;
}
```

```
}
```

the Dclass has to have : extends AbsCmdReq implements Neuronable
in this case upon the input : say x y times
it will generat an algorithm with one alg part (APSay) and send it to the noiron.

the APclass extends AbsAlgPart. here it simply says x within each cycle.
it also uses a custom marker I added (at) to remember how many times it happend
so as to stop after y times.

permission levels :
after you've built your skill you need to place the Dclass into the Chobit class c'tor methode
as such :
Code:

```
public Chobit() {
 super();
 noiron = new Neuron();
 this.inner = new InnerClass(); // sleep var
 DAlarmer dAlarmer = new DAlarmer();
 // add a skill here, only 1 line needed !!!
 dClassesLv1.add(new Detective(fusion));
 dClassesLv1.add(new DJirachi());
 dClassesLv1.add(new DIAutomatic(kokoro, master));
 dClassesLv1.add(new DHungry());
 dClassesLv1.add(dPermitter);
 dClassesLv1.add(new DRules((new APSleep(24)), inner));
 dClassesLv1.add(new DSpeller());
 dClassesLv1.add(new DCalculatorV1());
 dClassesLv1.add(dAlarmer);
 dClassesLv2.add(new DSayer());
 dClassesLv3.add(dAlarmer);
 dClassesLv3.add(new DDirtyTalker());
 // dClassesLv3.add(new DIMommyGf(kokoro, this.master));
 dClassesLv3.add(new DIJirachi(master, kokoro));
    }
```

lv1 : will run anyways
lv2 : requires lv1 permission : chobit name + input, for example :
chii say hi.
lv3 requires lv2 permission : chobit lv2 name + input, for example :
liron kiss me.
skills that engage by time triggers are defind as automatic if they are lv1 or higher

Code:

```
package chobit;
public abstract class AbsCmdReq implements Neuronable {
 // handle the input per Dclass (a class whose name starts with D)
 public abstract void input(String ear, String skin, String eye);

 public Boolean auto() {
 // does this skill also engage by time triggers ? is it also a level > 1 type of
 // skill ? if yes
 // override me and return true;
 return false;
 }
}
```

you simply add them as any other lv1 or 2 skill and they are automatically added to the auto skill list
in addition. this way they can be accessed by the user and by time to engage, but still restrict access to
any other person who wants to use the skill.

the DAlarmer is an example of such a skill. only the user can set an alarm, and the alarm is triggered by
time

but no one else can set an alarm.

the lv2 name will be known only to the owner, while her lv1 name is her public
name used by friends and owner.

of course don't forget to declare the Dclass variable within the chobit class.

unique features:

female titan : fusion of algorithms :

the LG remembers how long an alg run time is. and so short enough algs can pause
a running much longer alg, run themselfs, then resume the big alg(time wize).
you can add custom logic to this if needed via the :
fuze(){} in the Fusion class.

armored titan : ability to mutate an alg part.

refer to the APFilth1 and 2 classes as an exampled implementation of this.
said moan classes dont do much they simply output a moan string.

APFilth1 sends an enumFail.fail (action function)if no input is received x number of times :
in other words the user isn't enjoying this moan set.

and so the mutation causes the AP to be replaced by a newly generated AP from here :

make sure all the APs in the set have the same name + different number AND overide the getMutationLimit methode
within each of those AP classes of your mutation set, to the number of mutations the AP can have before the alg is rendered inActive.

Code:

```
if (failureCounter > 1) {
                        cera.setActive(false);
                    }
```

megata titan : dormant algorithms and its revival.
refer to DAlarmer and APCldAlarm for an example of an alarm clock using
this ability.
each Mcode or item has an equip algorithm (get the item, or declare it is equiped for
virtual items or do nothing)

an AP class as such uses a cloudian object and should be named with APCldSomething.

****
*****

extras :

the emot function of the chobit class links the AP running to an emotion
so this should be linked to graphics.

refer to classes javadoc for more info.

ideally the A.G.I shoulld be running offline, so preffer local device databases
over online ones, to keep the A.G.I "enjoyable".

****
*

examples of skills that need to be added :

auto programming : given task, outputs code project

titam level skill (war hammer titan): go home : when on stand by (finished doing stuff) and outside, go home but 1st collect items used. fix or upgrade part of item.

add vision input compatibility : visual data -> to string, add eye field to Dclasses and handle at AP classes. this fields can carry touch and other data strings.

limitless options really.

****
****

currently available conjurations :

copyrights moti barski

AP classes : extend the AbsAlgPart
those classes encapsulate the actual action or skill
APCuss : will cuss using the same word while the cause of the cuss is detected

APDirtyTalk : dirty talks with the user
conjuratis : "ok next" finishes the AP
c'tor : APDirtyTalk(Hashtable<String, String> hashtable)
gets a dirty talk DB as a dictionary. *that must start with "start"
further dev recommended : output filter : I U we they convert to U I and so on
advanced ver will learn new scripts, using vision and other techniques

APMoan : this is the moaning stage that comes after the dirty talk forplay AP in the algorithm.
a rather primitive moan system with preset moans :
moan0 to 2 = default moans, IMoan1 to 3 = moans when doll moved
thank you moan : finisher groan
private void playMoan(String input) {
is responsible for working the moan * could do this async

APSay : say x, y times

APSleep0 : sleeps : activates the private void sleep of the chobit class

APSleep : simulates sleep, until wake time or special wake input was inputed
*such input must undergo translation (noise = "wake up")and an if code line in the action function.

APSpell : can activate without permission(chobit name or hidden name)
such as telling times

D classes : extends the AbsCmdReq

this classes summon the AP classes using conjuratis(regex)

DDirtyTalker : dirty talk *
replace * with fuck
or program more options instead of *

Detective : this is a special DClass it doesn't use conjuratis
it is triggered by danger and negativity :
cussing

repeated input
too much requests
need to add : damage : body, friends, property, * time(cerabellum time out)

DPermitter : checks for the chobit name or lv 2 name before sending
a request to the other Dclasses.
sexy classes would require lv2 name
conjuratis :
change pass : oldPass new password newPassword change lv1 name :
pass your name is newName change lv2 name : pass you are newName

DRules : scheduled continuoes sync actions
contains sleep summon triggered by time

DSayer : says stuff
say x # times or say x

chobit : see class

SOUL SKILLS : DISKILL : CONSCIOUSNESS EQUIPPED SKILLS EXEMPLIFIED WITH THE USER IMPRINT SKILL.

this type of skill require the mendatory classes to exist in the project:
1 Bijuu : a container for a list of DISkills
2 Chi : an adaptor for AP classes which enables it to use the soul class
3 DISkill : a skill to be contained by a Bijuu class and generate conscious algorithms (of Chi parts).
4 kokoro : the AGIs soul, all data passes through it and it can do stuff as written in its class comments.

Bijuu

Code:

```
package chobit;

import java.util.ArrayList;

public class Bijuu extends AbsCmdReq {
 /*
 * a container cls for a list of DIskills
 */
 protected ArrayList<DISkill> dSkills = new ArrayList<>();
 private Kokoro kokoro;
 final int constTolerance = 3;
 private int tolerance = constTolerance;
 private Boolean enabled = true;
 private Boolean fastBreak = true;
 protected Person person;

 public Bijuu(Person master, Kokoro kokoro, DISkill... skills) {
 super();
 this.kokoro = kokoro;
 this.person = person;
 for (DISkill i : skills) {
 dSkills.add(i);
 }
 }

 public void modeFlip() {
 // pain = *repetition/ actual high level pain
 // sets weather the Bijuu is active or not
```

```
tolerance -= kokoro.getPain(this.getClass().getSimpleName());
if (tolerance < 1) {
this.enabled = !this.enabled;
}
}

@Override
public void input(String ear, String skin, String eye) {
if (enabled) {
// if Bijuu enabled
for (DISkill dISkill : dSkills) {
dISkill.input(ear, skin, eye);
if (dISkill.getSentAlg()) {
/*
* found an alg ! exit the loop ! I dont need another alg !!
*/
dISkill.setOutput(true);
// hey, DIskill, remind me you have an alg waiting for pickup
fastBreak = false;
// dont skip alg pick up stage.
break;
}
}
}
else {
reenable(ear, skin, eye); // maybe I should be revived
}
}

@Override
public void output(Neuron noiron) {
// TODO Auto-generated method stub
if (!fastBreak) {
// if alg waiting for pick up
fastBreak = true; // reset
for (DISkill dISkill : dSkills) {
if (dISkill.getOutput()) {
// found the alg
dISkill.output(noiron);
// OK done, bye
break;
}
}
}
}

public void reenable(String ear, String skin, String eye) {
if (ear.contains("pain") || skin.contains("pain")) {
tolerance -= 1;
if (tolerance < 1) {
this.enabled = true;
}
}
}
}
```

Chi :

Code:

```
package com.yotamarker.lgkotlin1;

public class Chi extends AbsAlgPart {
 /*
 * an adaptor pattern to the alg part, it also has the kokoro consiousness
 * object to be aware throughout the program of what is happening all action
 * data goes through this soul.
 */
 public Kokoro kokoro;
 public String ofSkill;
 public AbsAlgPart aPart;
```

```
public Chi(Kokoro kokoro, String ofSkill, AbsAlgPart aPart) {
super();
this.kokoro = kokoro;
this.ofSkill = ofSkill;
this.aPart = aPart;
}

public String actualAction(String ear, String skin, String eye) {
return aPart.action(ear, skin, eye);
}
@Override
public String action(String ear, String skin, String eye) {
kokoro.in(this);
String result = actualAction(ear, skin, eye);
kokoro.out(completed(), failure(""));
return result;
}

@Override
public Boolean itemize() {
// TODO Auto-generated method stub
return aPart.itemize();
}

@Override
public enumFail failure(String input) {
// TODO Auto-generated method stub
return aPart.failure(input);
}

@Override
public Boolean completed() {
// TODO Auto-generated method stub
return aPart.completed();
}

@Override
public AbsAlgPart clone() {
// TODO Auto-generated method stub
return new Chi(kokoro, this.ofSkill, aPart.clone());
}

}
```

DISkill

Code:

```
package chobit;

public class DISkill extends AbsCmdReq {
 protected Boolean sentAlg = false; // accessed by sub cls
 private Boolean output = false; // accessed by the DISkill container a Bijuu cls
 // String ofSkill;
 protected Kokoro kokoro; // accessed by sub cls

 public void setSentAlg(Boolean sentAlg) {
 this.sentAlg = sentAlg;
 }
 // in sub cls : person ?
 public DISkill(Kokoro kokoro) {
 super();
 // this.ofSkill = ofSkill;
 this.kokoro = kokoro;
 }
 @Override
 public void output(Neuron noiron) {
 // set sentAlg = true if an alg is to be sent

 }
```

```
 @Override
 public void input(String ear, String skin, String eye) {
 // TODO Auto-generated method stub

 }

 public Boolean getOutput() {
 Boolean result = this.output;
 this.output = false;
 return result;
 }

 public void setOutput(Boolean output) {
 this.output = output;
 }

 public Boolean getSentAlg() {
 Boolean result = this.sentAlg;
 this.sentAlg = false;
 return result;
 }
}
```

Kokoro :

Code:

```
package chobit;

import java.util.Hashtable;

/* all action data goes through here
 * detects negatives such as : repetition, pain on various levels and failures
 * serves as a database for memories, convos and alg generations
 * can trigger revenge algs
 * checks for % of difference in input for exploration type algs
 * */
public class Kokoro {
 Hashtable<String, Integer> pain = new Hashtable<>();

 public int getPain(String BijuuName) {
 return pain.getOrDefault(BijuuName, 0);
 }

 public void in(Chi chi) {
 }
 public void out(Boolean isCompleted, enumFail failure) {
 }
}
```

and the new upgraded Chobit class, with the kokoro field added :

Code:

```
package chobit;

import java.util.ArrayList;
import java.util.Hashtable;

public class Chobit {
   protected String emot = ""; // emotion
   protected ArrayList<AbsCmdReq> dClassesLv1 = new ArrayList<>();
   protected ArrayList<AbsCmdReq> dClassesLv2 = new ArrayList<>();
   protected ArrayList<AbsCmdReq> dClassesLv3 = new ArrayList<>();
   // algorithms fusion (polymarization)
   protected Hashtable<String, Integer> AlgDurations = new Hashtable<>();
   protected Fusion fusion = new Fusion(AlgDurations);
   // region essential DClasses
   protected Permission permission = Permission.newInstance("xxx", "chii", "liron");
```

```
  protected DPermitter dPermitter = new DPermitter(permission);
  // endregion
  protected Neuron noiron;
  // sleep vars :
  protected InnerClass inner;
  protected Person activePerson = new Person();
  protected PrimoCera primoCera = new PrimoCera();
  // added :
  protected Kokoro kokoro = new Kokoro(); // soul
  protected Person master = new Person();
   public Chobit() {
     super();
     noiron = new Neuron();
     this.inner = new InnerClass(); // sleep var
     DAlarmer dAlarmer = new DAlarmer();
     // add a skill here, only 1 line needed !!!
     dClassesLv1.add(new Detective(fusion));
     dClassesLv1.add(new DJirachi());

     dClassesLv1.add(new DHungry());
     dClassesLv1.add(dPermitter);
     dClassesLv1.add(new DRules((new APSleep(24)), inner));
     dClassesLv1.add(new DSpeller());
     dClassesLv1.add(new DCalculatorV1());
     dClassesLv1.add(dAlarmer);
     dClassesLv2.add(new DSayer());
     dClassesLv3.add(dAlarmer);
     dClassesLv3.add(new DDirtyTalker());
     // dClassesLv3.add(new DIMommyGf(kokoro, this.master));
     dClassesLv3.add(new DIJirachi(master, kokoro));
   }

  public String doIt(String ear, String skin, String eye) {
     for (AbsCmdReq dCls : dClassesLv1) {
        inOut(dCls, ear, skin, eye);
     }
     if (dPermitter.getPermissionLevel() > 0) {
        // works with friends
        for (AbsCmdReq dCls : dClassesLv2) {
           inOut(dCls, ear, skin, eye);

        }
     }
     if (dPermitter.getPermissionLevel() > 1) {
        // only works with owner
        for (AbsCmdReq dCls : dClassesLv3) {
           inOut(dCls, ear, skin, eye);
        }
     }
     fusion.setAlgQueue(noiron);
     return fusion.act(ear, skin, eye);
  }

   public String getEmot() {
     // emot (emotion for display)
       String x1 = emot;
       switch (this.emot) {
           case "APCuss ":
               x1 = "angry";
               break;
           case "APDirtyTalk":
               x1 = "grinny";
               break;
           case "APMoan":
               x1 = "horny";
               break;
           case "APSay":
               x1 = "speaking";
               break;
           case "APSleep0":
```

```
                x1 = "dreaming";
                break;
            case "APSleep":
                x1 = "asleep";
                break;
            case "APSpell":
                x1 = "blank";
                break;
            default:
                break;
        }
        emot = "";
        return x1;
    }

   protected String sleep() {
      // data save load should go here and run while chobit is sleeping
        return "haha I can sleep !";
    }

   protected void inOut(AbsCmdReq dClass, String ear, String skin, String eye) {
      dClass.input(ear, skin, eye); // new
        dClass.output(noiron);
    }

    protected class InnerClass {
        public String nemure() {
            return sleep();
        }
    }
   protected String translateIn() {
      return "";
   }

   protected String translateOut() {
      return "";
   }
}
```

and also the Person class added to the Chobit class :

Code:

```
package chobit;

public class Person {
 private String name = "";
 private Boolean active = true;
 private String phone = "";
 private String skill = "";
 private String profession = "";
 private String jutsu = "";
 // location
 private String email = "";
 private String id = "";
 public String getName() {
 return name;
 }
 public void setName(String name) {
 this.name = name;
 }
 public Boolean getActive() {
 return active;
 }
 public void setActive(Boolean active) {
 this.active = active;
 }
 public String getPhone() {
 return phone;
 }
 public void setPhone(String phone) {
```

```
this.phone = phone;
}

public String getSkill() {
return skill;
}

public void setSkill(String skill) {
this.skill = skill;
}

public String getProfession() {
return profession;
}

public void setProfession(String profession) {
this.profession = profession;
}

public String getJutsu() {
return jutsu;
}

public void setJutsu(String jutsu) {
this.jutsu = jutsu;
}

public String getEmail() {
return email;
}

public void setEmail(String email) {
this.email = email;
}

 public String getId() {
 return id;
 }

 public void setId(String id) {
 this.id = id;
 }
}
```

this classes names start with DI

the action part : APImprintMaster :

Code:

```
package com.yotamarker.lgkotlin1;

import java.util.ArrayList;

public class APImprintMaster extends AbsAlgPart {
   // todo : handle inputs regexes
```

```
   /*
    * asks master for vital info, to fill in master object fields like name and
    * phone number
    */
   private ArrayList<String> form = new ArrayList<>();
   private int mode = 0;
   private int index = 0;
   private String input = "";
   private Boolean isCompleted = false;
   private String curResult = "";
   private Person master;

   public APImprintMaster(Person master) {
      // default c'tor
      super();
      form.add("ntmyawwynb");
      form.add("are you my master");
      form.add("I will input your name");
      form.add("what is your skill");
      form.add("what is your profession");
      form.add("what is your phone number");
      form.add("what is your email address");
      form.add("which is your favorite jutsu");
      form.add("soul spark engaged");
      this.master = master;
   }

   public APImprintMaster(Person master, String... strs) {
      // alternative c'tor
      for (String i : strs) {
         form.add(i);
      }
      this.master = master;
   }
```

```
   @Override
   public String action(String ear, String skin, String eye) {
      String result = "";
      switch (mode) {
      case 0:
         result = form.get(index);
         curResult = result;
         if (result.contains("what") || result.contains("which is") ||
result.contains("ntmyawwynb") || result.contains("please")
               || result.contains("are you")) {
            mode = 2;
            if (form.get(index).contains("are you")) {
               mode = 4;
            }
         }
         else {
            index++;
         }
         break;
      case 2:
         if (!ear.isEmpty() && !form.contains(ear)) {
            mode = 3;
            input = ear;
         }

         break;
      case 3:
         result = input + " yes";
         mode = 4;
         break;
      case 4:
         if (ear.contains("yes")) {
            mode = 0;
            imprint();
            index++;
```

```
         }
         if (ear.contains("no")) {
            mode = 0;
         }
         break;
      default:
         break;
      }
      if (index == form.size()) {
         isCompleted = true;
         index--;
      }
      return result;
   }

   public void imprint() {
      switch (curResult) {
      case "nice to meet you, and what would your name be":
         master.setName(input);
         break;
      case "what is your skill":
         master.setSkill(input);
         break;
      case "what is your profession":
         master.setProfession(input);
         break;
      case "what is your phone number":
         master.setPhone(input);
         break;
      case "what is your email address":
         master.setEmail(input);
         break;
      case "which is your favorite jutsu":
         master.setJutsu(input);
         break;
      default:
         break;
      }
   }

   @Override
   public Boolean itemize() {
      // TODO Auto-generated method stub
      return false;
   }

   @Override
   public enumFail failure(String input) {
      // TODO Auto-generated method stub
      return enumFail.ok;
   }

   @Override
   public Boolean completed() {
      // TODO Auto-generated method stub
      return isCompleted;
   }

   @Override
   public AbsAlgPart clone() {
      // ***might glich, clone person ?
      return new APImprintMaster(this.master);
   }

}
```

DIMommyGF class extends DISkill:

Code:

```
package chobit;
```

```
import java.util.ArrayList;

public class DIMommyGf extends DISkill {
   public Person master;
   private Boolean exeAlg;

   public DIMommyGf(Kokoro kokoro, Person owner) {
      super(kokoro);
      this.master = owner;
      // TODO Auto-generated constructor stub
   }

   @Override
   public void input(String ear, String skin, String eye) {
      if (ear.contains("imprint master")) {
         this.exeAlg = true;
         this.setSentAlg(true);
      }
   }

   @Override
   public void output(Neuron noiron) {
      // TODO Auto-generated method stub

      if (this.exeAlg) {
         AbsAlgPart itte = new Chi(this.kokoro, this.getClass().getSimpleName(), new APImprintMas-
ter(this.master));
         String representation = "imprintmaster";
         ArrayList<AbsAlgPart> algParts1 = new ArrayList<>();
         algParts1.add(itte);
         Algorithm algorithm = new Algorithm("imprintmaster", representation, algParts1);
         noiron.algParts.add(algorithm);
         exeAlg = false;
      }
   }
}
```

DIJirachi class :

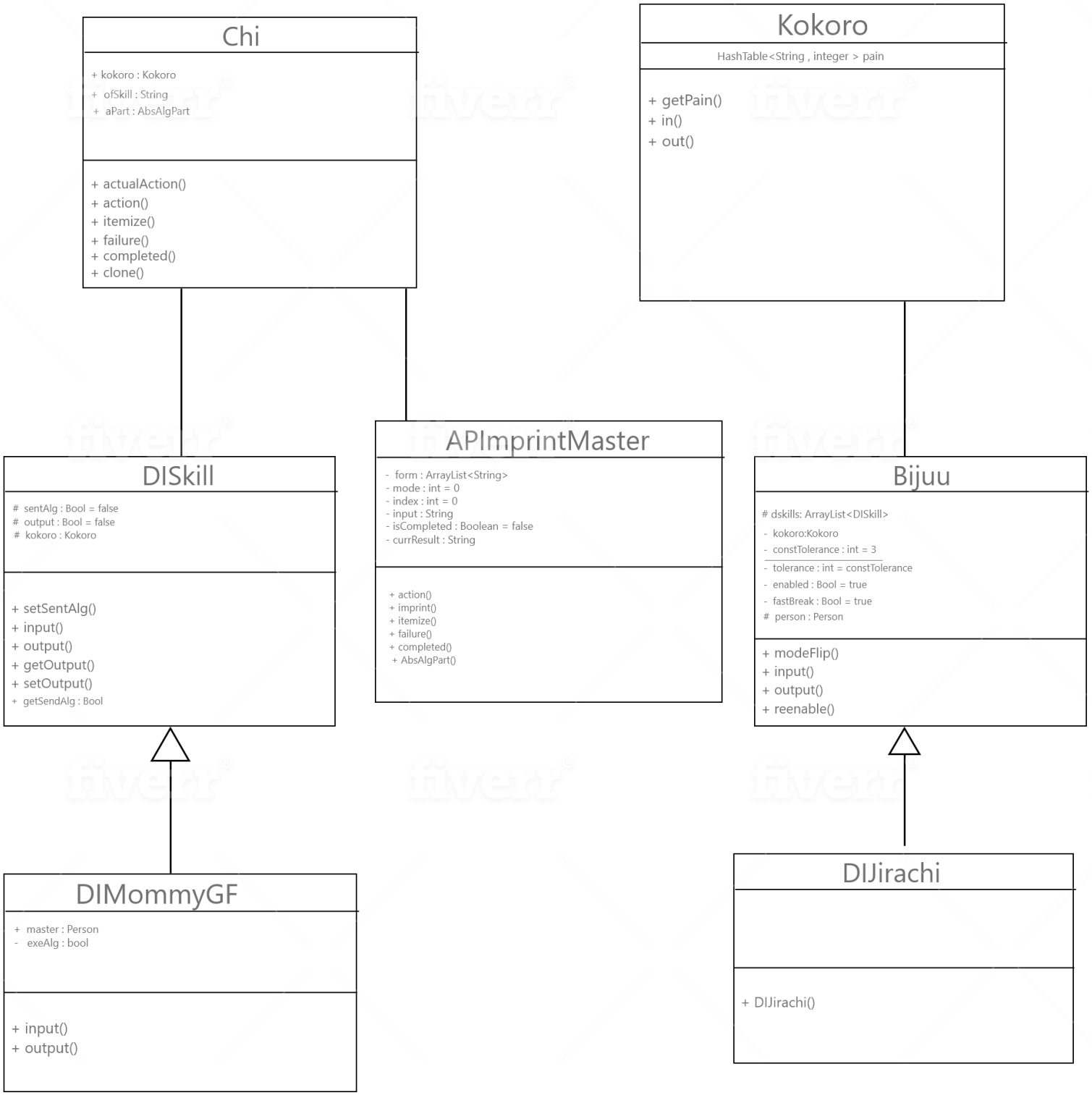
Code:

```
package chobit;

public class DIJirachi extends Bijuu {
   // a set of skills to make the user happy and grant his wishes
   public DIJirachi(Person master, Kokoro kokoro) {
      super(master, kokoro, new DIMommyGf(kokoro, master));

   }

}
```

DIJirachi is the actual skill here and it is added in the Chobit class constructor (using one code line)

THE ESSENCIAL SKILLS AN AI SHOULD HAVE:

Subject: A.G.I juubi Sat Oct 05, 2019 3:02 pm	quote edit X ?


Chi
+ kokoro : Kokoro
+ ofSkill : String
+ aPart : AbsAlgPart
+ actualAction()
+ action()
+ itemize()
+ failure()
+ completed()
+ clone()
Kokoro
HashTable<String , integer > pain
+ getPain()
+ in()
+ out()
DISkill
sentAlg : Bool = false
output : Bool = false
kokoro : Kokoro
+ setSentAlg()
+ input()
+ output()
+ getOutput()
+ setOutput()
+ getSendAlg : Bool
APImprintMaster
- form : ArrayList<String>
- mode : int = 0
- index : int = 0
- input : String
- isCompleted : Boolean = false
- currResult : String
+ action()
+ imprint()
+ itemize()
+ failure()
+ completed()
+ AbsAlgPart()
Bijuu
dskills: ArrayList<DISkill>
- kokoro:Kokoro
- constTolerance : int = 3
- tolerance : int = constTolerance
- enabled : Bool = true
- fastBreak : Bool = true
person : Person
+ modeFlip()
+ input()
+ output()
+ reenable()
DIMommyGF
+ master : Person
- exeAlg : bool
+ input()
+ output()
DIJirachi
+ DIJirachi()


A.G.I
JUUBI MODE
imgflip.com

To sum things up :
You use one code line to add a skill to the cohabit
You use the doIt function of said cohabit to have it output the needed result passing it
Ear skin eye data as strings.

Indeed this is the most efficient way to code.

3RD GENERATION OF SKILLS FOR THE AGI PLATFORM

TheSkill (java), skill names of this type of TheSkill should start with The

Code:

```
package chobit;

import java.util.ArrayList;

public abstract class TheSkill extends DISkill {
   protected RegexUtil regexUtil = new RegexUtil();
   protected DISkillUtils diSkillUtil = new DISkillUtils();
   protected PlayGround playGround = new PlayGround();
   protected CloudianV2 cloudian = new CloudianV2();
   private MCodes mCodes = null; // items

   public void setmCodes(MCodes mCodes) {
      this.mCodes = mCodes;
   }

   public void setFriend(Person friend) {
      this.friend = friend;
   }

   public void setAbsDefCon(AbsDefconV2 absDefCon) {
      this.absDefCon = absDefCon;
   }

   private SupeReplikaMap replikaMap = new SupeReplikaMap();
   protected Person friend = null; // if you deal with several friends handle it in the sub class
   private Boolean friendUpdatable = false;
   private ArrayList<String> items = new ArrayList<String>();
   private String item = "";
   protected AbsDefconV2 absDefCon;
   protected Algorithm outputAlg = null;
   private String clsName = "*(^^%&*";

   public void setItems(ArrayList<String> items) {
      this.items = items;
   }

   // public TheSkill(Kokoro kokoro, AbsDefconV2 absDefCon, ArrayList<String>
   // items) {
   // super(kokoro);
   // this.items = items;
   // this.absDefCon = absDefCon;
   // }

   public TheSkill(Kokoro kokoro, AbsDefconV2 absDefCon, ArrayList<String> items, String clsName) {
```

```
      super(kokoro);
      this.items = items;
      this.absDefCon = absDefCon;
      this.clsName = clsName;
   }
   @Override
   public void input(String ear, String skin, String eye) {
      detectFriend(ear);// title refet to code for elab
      // func1 :
      this.outputAlg = this.absDefCon.getDefcon(ear, skin, eye);// detects and handles custom
threats in the context
      // of a TheSkill
      inputToSoul(ear, skin, eye);// refer to methode for elab
      if (outputAlg != null) {
         return;
      }
      // func2
      triggeredAlgs(ear, skin, eye);
      if (!isNull(outputAlg)) {
         return;
      }
      // func3
      this.outputAlg = soulOutput(ear);
      /*
       * answers questions related to skill items. add friend to item list to get
       * friend info as well
       */
   }

   private void triggeredAlgs(String ear, String skin, String eye) {
      trgAction(ear, skin, eye);// actions triggered by input
      if (!isNull(outputAlg)) {
         return;
      }
      trgExplore(ear, skin, eye);// time triggered actions should go here
      if (!isNull(outputAlg)) {
         return;
      }
      trgPreserve(ear, skin, eye);// actions related to getting a friend or maintaining items help-
ful for the
                           // skill goals
      // if they aren't helpful an algorithm part AP class should delete them from the
      // skill so the skill is free to
      // make a new friend or get a better item
   }

   @Override
   public void output(Neuron noiron) {
      if (!isNull(this.outputAlg)) {
         noiron.algParts.add(this.outputAlg);
         this.outputAlg = null;
      }
      // after this, if there is no reference to the object,
      // it will be deleted by the garbage collector
   }

   private boolean isNull(Object obj) {
      return obj == null;
   }

   protected abstract void trgAction(String ear, String skin, String eye);
   // sensory, souled(kokoro cls directives), predicted

   protected abstract void trgExplore(String ear, String skin, String eye);

   // timed
   // Exploration and learning, Alg efficiancy tests and sort
   protected abstract void trgPreserve(String ear, String skin, String eye);
   // timed
   // items and persons preservation, causes being annoyed if repeated in day
```

```
   protected Algorithm makeFriend() {
      return diSkillUtil.verbatimGorithm(new APVerbatim("what is your name"));
   }

   protected void friendUpdate(String ear) {
      String temp = regexUtil.phoneRegex1(ear);
      if (!temp.isEmpty()) {
         friend.setPhone(temp);
      }
      temp = regexUtil.emailRegex(ear);
      if (!temp.isEmpty()) {
         friend.setEmail(temp);
      }
      temp = regexUtil.afterWord("i am ", ear);
      if (temp.isEmpty()) {
         temp = regexUtil.afterWord("my name is ", ear);
      }
      if (!temp.isEmpty()) {
         friend.setName(temp);
         friend.setActive(true);
      }
      temp = regexUtil.duplicateRegex(ear);
      if (!temp.isEmpty()) {
         friend.setJutsu(temp);
         friend.setActive(true);
      }
   }

   // key stuff detection and handling
   protected void detectFriend(String ear) {
      if (playGround.getMinutesAsInt() % 2 == 0) {
         friendUpdatable = false;
      }
      Boolean friendRequest = (ear.contains("friends") || ear.contains("my name is")) && !
this.friend.getActive();
      // a friend is set to false and cleared on algPart failures
      if (friendRequest) {
         // at this case a friend volunteers himself.
         kokoro.toHeart.put("Me", "introduce");// this can be summoned in the trgPreserve in case
         // no friend.
      }
      if (ear.contains(friend.getName()) || (ear.contains(friend.getJutsu())) || friendRequest)//
or friend visual
      {
         friendUpdatable = true;
         if (items.contains("friend")) {
            this.item = "friend";
         }
         // friend patch
         // the friend is active and therefore can update his info
      }
      if (friendUpdatable) {
         friendUpdate(ear);
      }
   }

   protected String currentItem(String ear, String skin, String eye) {
      for (String item : items) {
         if (eye.contains(item)) {
            return item;
         }
      }
      for (String item : items) {
         if (skin.contains(item)) {
            return item;
         }
      }
      for (String item : items) {
         if (ear.contains(item)) {
```

```
            return item;
         }
      }
      return "";
   }

   public static String strContains(String str1, String... a) {
      for (String temp : a) {
         if (str1.contains(temp)) {
            return temp;
         }
      }
      return "";
   }

   public static String strContainsList(String str1, ArrayList<String> items) {
      for (String temp : items) {
         if (str1.contains(temp)) {
            return temp;
         }
      }
      return "";
   }

   protected void inputToSoul(String ear, String skin, String eye) {
      /*
       * in the condition the bot is moving (static class compass is active) the
       * skills map will record info related to items or threats pertaining to a
       * specific TheSkill mainly sensory input, and a detected threat specific to the
       * TheSkill and defined in it's AbsDefconVs object
       */
      String sensory = ear;
      String currentDefcon = this.absDefCon.getAbsoluteDefcon(ear, skin, eye);
      if (sensory.isEmpty()) {
         sensory = skin;
      }
      if (sensory.isEmpty()) {
         sensory = eye;
      }
      if (!item.equals("friend")) {
         this.item = currentItem(ear, skin, eye);
      } // friend patch
      if (!this.item.isEmpty() || !currentDefcon.isEmpty()) {
         this.replikaMap.input(item, currentDefcon, sensory);
         this.item = "";
      }
   }

   protected Algorithm soulOutput(String ear) // ear or time
   {
      String question = strContains(ear, "what", "describe", "where");
      switch (question) {
      case "where":
         String tempItem = strContainsList(ear, items);// gets item in question
         return diSkillUtil.verbatimGorithm(new APVerbatim(replikaMap.where(tempItem)));
      default:
         if (!question.isEmpty() && ear.contains(this.clsName)) {
            return diSkillUtil.verbatimGorithm(new APVerbatim(replikaMap.answer(question)));
         }
         break;
      }
      return null;
   }

   protected Algorithm absorbedSkill(String ear, String skin, String eye, DISkill oldSkill) {
      // an adapter for upgrading DISkills to TheSkills
      Neuron tempNeuron = new Neuron();
      oldSkill.input(ear, skin, eye);
      oldSkill.output(tempNeuron);
      if (!tempNeuron.algParts.isEmpty()) {
```

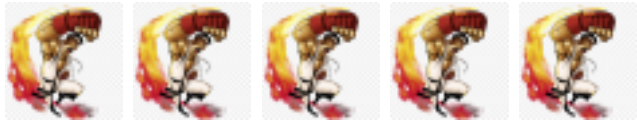
```
            return tempNeuron.algParts.get(0);
         }
         return null;
      }
   }
```

example subclass : TheSitter

Code:

```
package chobit;

import java.util.ArrayList;

public class TheSitter extends TheSkill {
   /*
    * prayer1 : Thank You, God, for Loving Me! The sun is setting, dear Father, and
    * it's time to go to bed. Thank you, God, for loving me from my toesies to my
    * head. Tomorrow will be another fun and silly and busy day. Please keep me
    * safe as I run and jump and giggle and sing and play.
    */
   private DISitter diSitter = null;
   private InstaConvo instaConvo = new InstaConvo();
   // tick trigger values :
   private String sent1 = "";
   private int algMode = 0;
   private ZeroTimeGate tGSitter = new ZeroTimeGate(0);
   // end tick triggers
   public TheSitter(Kokoro kokoro, String clsName) {
      super(kokoro, null, null, clsName);
      ArrayList<String> items = new ArrayList<String>();
      items.add("pacifier");
      items.add("diaper");
      Person friend = new Person();
      MCodes mCodes = new MCodes();
      AbsDefconV2 absDefconV2 = new ABDLDefcon(mCodes, friend);
      this.setFriend(friend);
      this.setmCodes(mCodes);
      this.setAbsDefCon(absDefconV2);
      this.setItems(items);
      // absorbtion of old skill :
      this.diSitter = new DISitter(kokoro);
      // insta convos set up
      instaConvo.loadBullet("i love you", "i love you too").loadBullet("i love you", "really", "of
course kido");
      instaConvo.loadBullet("what is your objective", "to nurse and protect");
   }

   @Override
   protected void trgAction(String ear, String skin, String eye) {
      // absorb old diskill capabilities
      outputAlg = absorbedSkill(ear, skin, eye, diSitter);
      if (outputAlg != null) {
         return;
      }
      if (this.friend.getActive()) {
      String ic1 = instaConvo.converse(ear.toLowerCase());
      if (!ic1.isEmpty()) {
         outputAlg = diSkillUtil.verbatimGorithm(new APVerbatim(ic1));
         return;
         }
      }
   }
   @Override
   protected void trgExplore(String ear, String skin, String eye) {
      String now = playGround.getCurrentTimeStamp();
      if (!sent1.equals(now)) {
         sent1 = "";
         switch (now) {
```

```
         case "05:00":
            if (playGround.getMonthAsInt() == 1) {
               this.friend.deleteFriend();
            }
            sent1 = "05:00";
            return;
         case "01:00":
            outputAlg = diSkillUtil.verbatimGorithm(new APVerbatim("filthx1"));// masturbatin
            sent1 = "01:00";
            algMode = 1;// context
            tGSitter.open(50);
            return;
         case "19:05":
            sent1 = "19:05";
            outputAlg = diSkillUtil.verbatimGorithm(new APVerbatim("prayer1"));
            return;
         default:
            break;
         }
         // end if
      }
      switch (algMode) {
      case 1:
         if (tGSitter.isClosed() || ear.contains("no") || ear.contains("problem")) {
            algMode = 0;
         }
         if (ear.contains("mommy")) {
            outputAlg = diSkillUtil
                  .verbatimGorithm(
                        new APVerbatim("bad boy interupting", "go stand in the corner for 10 min-
utes"));
            return;
         }
         break;
      default:
         break;
      }
   }

   @Override
   protected void trgPreserve(String ear, String skin, String eye) {


   }

   @Override
   public Boolean auto() {
      return true;
   }
}
```



TOGGLER TYPE SKILLS FOR THE LIVING GRIMOIRE AGI PLATFORM

to use this type of skills 2 lines of codes are required in the chobit class c'tor
adding a lv1 skill : with super class : DiTglrAdapter containing the desired skill

and adding a DiTglrSkill preferably as a lv3 skill. refer to class doc for more

Code:
```
package chobit;

public class DiTglrAdapter extends DISkill {
   // adapts a skill to be toggled on or off be a DiTglrSkill
   // this skills should be added only as level1 skills, example in the chobit
   // c'tor:
   // dClassesLv1.add(new DiTglrAdptrMommy(kokoro));
   private Boolean alive = true;
   private String conjuration = "";
   private DISkill diSkill;
   public DiTglrAdapter(Kokoro kokoro, String conjuration, DISkill diSkill) {
      super(kokoro);
      this.diSkill = diSkill;
      this.conjuration = conjuration;
   }

   public DiTglrAdapter(Boolean startAsActive, Kokoro kokoro, String conjuration, DISkill diSkill)
{
      super(kokoro);
      this.diSkill = diSkill;
      this.conjuration = conjuration;
      this.alive = startAsActive;
   }
   @Override
   public void input(String ear, String skin, String eye) {
      // toggle :
      String meirei = this.kokoro.toHeart.getOrDefault(conjuration, "");
      if (meirei.contains(conjuration + " off")) {
         this.alive = false;
         this.kokoro.toHeart.remove(conjuration);
         return;
      }
      if (meirei.contains(conjuration + " on")) {
         this.alive = true;
         this.kokoro.toHeart.remove(conjuration);
         return;
      }
      // engage :
      if (alive) {
         this.diSkill.input(ear, skin, eye);
      }
   }

   @Override
   public void output(Neuron noiron) {
      if (alive) {
         this.diSkill.output(noiron);
      }
   }

}
```

DiTglrSkill :

Code:
```
package chobit;

import java.util.ArrayList;

public class DiTglrSkill extends DISkill {
   // this toggles a skill, logically a level 1 skill
   /*
    * thus one can use hidden skills without the chobits hidden name and use select
    * cases rather than string.contains for a speed beef up as well
```

```
    */
   private ArrayList<String> conjurtions = new ArrayList<String>();
   private String outString = "";
   private DISkillUtils diSkillUtils = new DISkillUtils();

   public DiTglrSkill(Kokoro kokoro, String... conjurationsTemp) {
      super(kokoro);
      for (int i = 0; i < conjurationsTemp.length; i++) {
         this.conjurtions.add(conjurationsTemp[i]);
      }
   }

   private String strContainsList(String str1) {
      for (String temp : conjurtions) {
         if (str1.contains(temp)) {
            return temp;
         }
      }
      return "";
   }
   @Override
   public void input(String ear, String skin, String eye) {
      // toggle :
      if (ear.contains("dislike")) {
         String conjurati = strContainsList(ear);
         if (!conjurati.isEmpty()) {
            kokoro.toHeart.put(conjurati, conjurati + " off");
            outString = "you dislike it";
            return;
         }
      }
      if (ear.contains("like")) {
         String conjurati = strContainsList(ear);
         if (!conjurati.isEmpty()) {
            kokoro.toHeart.put(conjurati, conjurati + " on");
            outString = "ok";
            return;
         }
      }
   }

   @Override
   public void output(Neuron noiron) {
      if (!outString.isEmpty()) {
         noiron.algParts.add(diSkillUtils.verbatimGorithm(new APVerbatim(outString)));
         outString = "";
      }
   }
}
```