MOTI BARSKI

THE LIVING GRIMOIRE

1ST EDITION

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

I moti barski do not allow anyone and or anybody and or any organization to
receive monetary profit from this living grimoire unless writtenly approved by me personally.
you can use this for research.

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 :

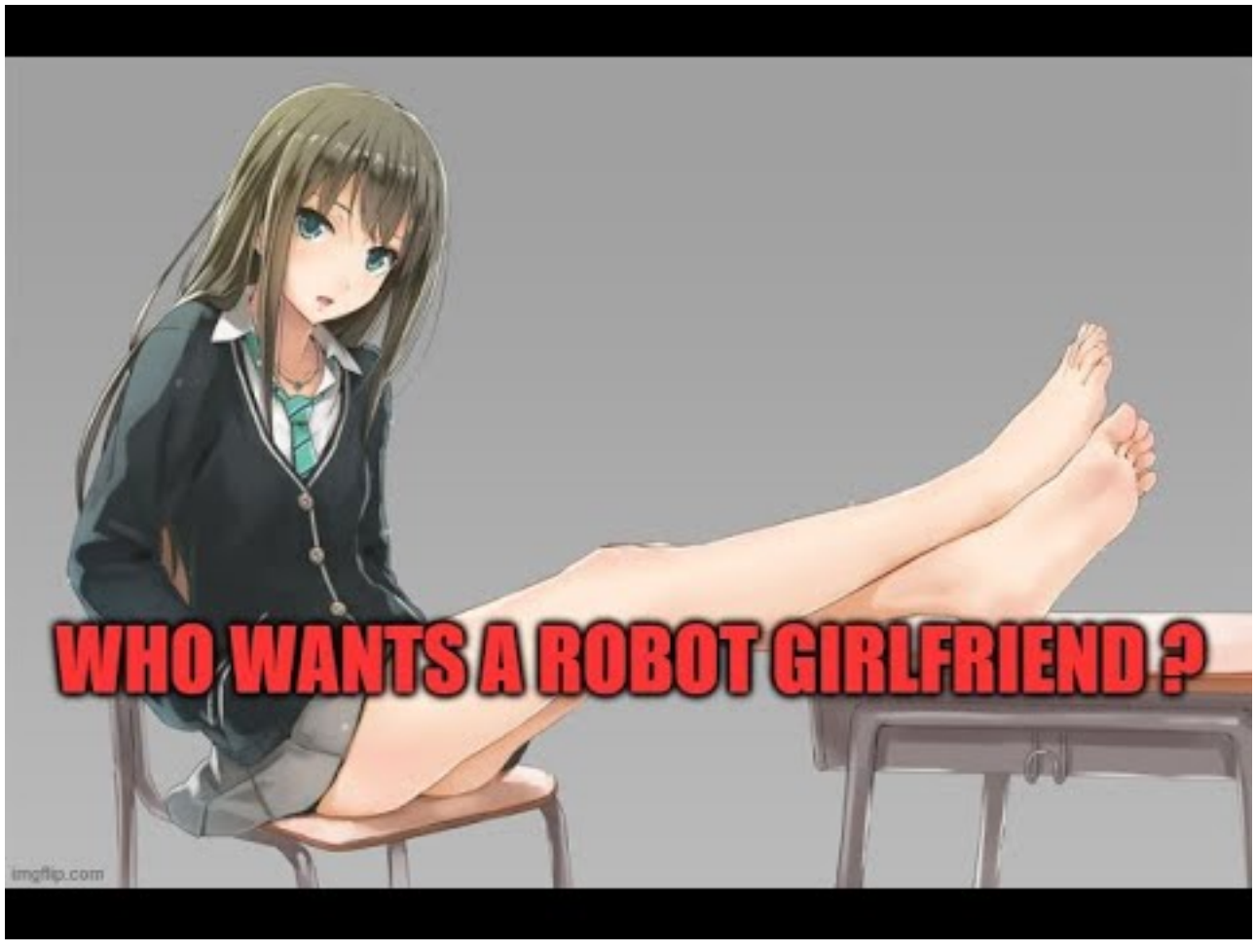
https://github.com/yotamarker/public-livingrimoire

FORUM FOR LIVING GRIMOIRE SKILLS :

http://livingrimoire.com/forum/

PRINCIPLE OF WORK :

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



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

what is a skill ?
a skill consists 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 should be running offline, so prefer 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

titan 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.

****
****

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

this type of skill extends DiSkill:
1 DISkill : a skill that has a reference to the kokoro class aka a soul skill
2 Kokoro : the AGIs soul,

the kokoro class is simply a shadow reference class present in all DiSkills and TheSkills thus enabling the skills to communicate between each other.
more over the kokoro class is in charge of saving the last mutated state of algParts (this is automatic and does not require any additional coding)

```
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 {
    private String emot = "";

    public String getEmot() {
        return emot;
    }
```

```
    public void setEmot(String emot) {
        this.emot = emot;
    }

    Hashtable<String, Integer> pain = new Hashtable<>();
      public GrimoireMemento grimoireMemento;
      public Hashtable<String, String> toHeart = new Hashtable<>();
      public Hashtable<String, String> fromHeart = new Hashtable<>();
      public Boolean standBy = false;
      public Kokoro(AbsDictionaryDB absDictionaryDB) {
            super();
            this.grimoireMemento = new GrimoireMemento(absDictionaryDB);
      }

      public int getPain(String BijuuName) {
            return pain.getOrDefault(BijuuName, 0);
      }
      public void out(Boolean isCompleted, enumFail failure) {
      }
}
```

DiSkill classes names start with DI. an example is :

makes the AGI burp, can be applied for belches or other bio sounds

```
package chobit;

import java.util.ArrayList;
import java.util.Random;

public class DIBurper extends DISkill {
   ArrayList<Integer> minutesToBurp = new ArrayList<Integer>();
   private PlayGround playGround = new PlayGround();
   private Random randomGenerator = new Random();
   private Boolean algToGo = false;
   public DIBurper(Kokoro kokoro) {
      super(kokoro);
      minutesToBurp.clear();
      int randomInt = randomGenerator.nextInt(60) + 1; // how many burps this hour
      for (int i = 0; i < randomInt; i++) {
         randomInt = randomGenerator.nextInt(60) + 1; // burp minute, add x random burps
         if (!minutesToBurp.contains(randomInt)) {
            minutesToBurp.add(randomInt);
         }
      }
   }

   @Override
   public void input(String ear, String skin, String eye) {
      int minutes = playGround.getMinutesAsInt();
      if (minutes == 0) {
         minutesToBurp.clear();
         int randomInt = randomGenerator.nextInt(60) + 1; // how many burps this hour
         for (int i = 0; i < randomInt; i++) {
            randomInt = randomGenerator.nextInt(60) + 1; // burp minute, add x random burps
            if (!minutesToBurp.contains(randomInt)) {
               minutesToBurp.add(randomInt);
```

```
            }
         }
      } else {
         if (minutesToBurp.contains(minutes)) {
            algToGo = true;
            this.setSentAlg(true);
         }
      }
   }

   @Override
   public void output(Neuron noiron) {
      if (algToGo) {
         algToGo = false;
         noiron.algParts.add(burp());
      }
   }

   private Algorithm burp() {
      AbsAlgPart itte = new Chi(this.kokoro, this.getClass().getSimpleName(), new APSay(1, "burp"));
      String representation = "burp";
      ArrayList<AbsAlgPart> algParts1 = new ArrayList<>();
      algParts1.add(itte);
      Algorithm algorithm = new Algorithm("burp", representation, algParts1);
      return algorithm;
   }
   @Override
   public Boolean auto() {
      // TODO Auto-generated method stub
      return true;
   }
}
```

THE ESSENCIAL SKILLS AN AI SHOULD HAVE:

DI skills an AGI needs mostly :

1 Hungry : for eating or charging , sleep
2 wishgranter : for pleasing the user (dirty stuff, caring, ...)
3 protection and self preservation
4 Work : for working
5 Programming
6 Homer : going home at the end of completing a goal or mission
7 Gamer
8 Breeder : recreating her self, and understanding her own algorithms and how to build another one
9 DISoul : memories, convos and alg generations
trigger revenge algs, recognize and avoid boredom

while the above are skills equipped with a soul and consciousness there can also be little skills that don't require a soul like :

permission skill : for enabling dirty stuff with the user, and only working not for free, stuff like that

therefore the AGI platform is the gedomazo while the skills are Bijuus ichiibii to kyubi to form the juubi

To sum things up :
You use one code line to add a skill to the Chobit class
You use the doIt function of said Chobit,passing it
Ear skin eye data as strings, to have it output the needed result
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
this are like DiSkills but with a map, containing sensory and danger memories per point

```
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
helpful 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) {
```

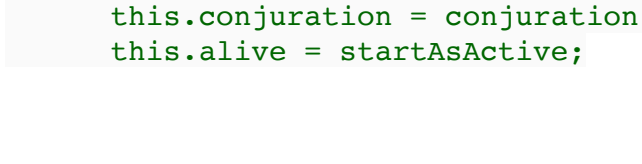
```
      /*
       * in the condition the bot is moving (static class compass is active) the
       * skills map will record info related to items or threats pertaining to a
       * specific TheSkill mainly sensory input, and a detected threat specific to the
       * TheSkill and defined in it's AbsDefconVs 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

```
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
minutes"));
            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. the later toggles the 1st skill on or off

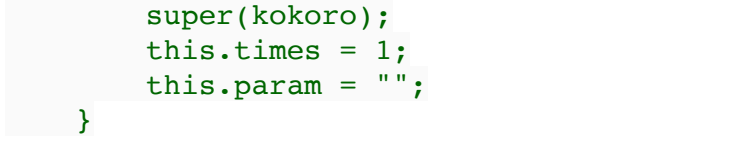
```
package chobit;

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 :

```
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 = "";
        }
    }
}
```



USING DATABASE FUNCTION :

the GrimoireMemento class of the Kokoro class
has a save and a load function.

any new skill that uses the kokoro class such as DiSkill and TheSkill can use DB capabilities to save and load.

her is the 1st example use : DiSayer class (java cls)
example use : honey say pen
output : pen

honey say something
output : pen
it doesn't matter if you turned off the app and reopened. she remembers.

```
package com.yotamarker.lgkotlin1;

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

public class DiSayer extends  DISkill{
    private int times;
    private String param;
    public DiSayer(Kokoro kokoro) {
        super(kokoro);
        this.times = 1;
        this.param = "";
    }

    @Override
    public void input(String ear, String skin, String eye) {
        if(ear.contains("say something")){
            this.param = kokoro.grimoireMemento.simpleLoad("something");
            return;
```

```
        }
        int foo = 1;
        String myString = regexChecker("(\\d+)(?= times)", ear);
        String toSay = regexChecker("(?<=say)(.*)(?=\\d)", ear);
        if (myString != "") {
            foo = Integer.parseInt(myString);
        } else {
            toSay = regexChecker("(?<=say)(.*)", ear);
        }
        this.param = toSay;
        this.times = foo;
    }

    @Override
    public void output(Neuron noiron) {
        if (!param.isEmpty()) {
            this.kokoro.grimoireMemento.simpleSave("something",param);
            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);
        }
    }
    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 "";
        }
    }
```

CHOBITV2 PERSONALITY UPGRADE

I am a lazy coder
which is why I really like the concept of one line of code to add a skill to the waifubot AGI.
but I understand I was not lazy enough.
indeed this is the most efficient way to code, and it offers customization, flexability and speed but overall a waifubot has several skills in total, one could find himself writing 9 maybe even 20 lines of code to fully load a bots personality.

what more I was thinking about gaming. some games would require many different AIs with different personality.

so, I beefed up the my waifubots c'tor !
1 line of code to add an entire personality(skill set) of the coders choice. you can not get lazier than that. that's like chew my food lazy.

this class is used in the ChobitV2 c'tor.
it enables loading a complete skill set (a sub class of the personality class)

using 1 line of code. of course you can also select specific skills to add from
the subclasses c'tor. see also Personality1 for example.

Personality :

Code:

```
package com.yotamarker.lgkotlin1;

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

public class Personality {
    /*this class is used in the ChobitV2 c'tor.
it enables loading a complete skill set (a sub class of the personality class)
using 1 line of code. of course you can also select specific skills to add from
the subclasses c'tor. see also Personality1 for example.*/
    protected Kokoro kokoro; // soul
    protected ArrayList<AbsCmdReq> dClassesLv1 = new ArrayList<>();// can engage with anyone
    protected ArrayList<AbsCmdReq> dClassesLv2 = new ArrayList<>();// can engage with friends and
work related
    protected ArrayList<AbsCmdReq> dClassesLv3 = new ArrayList<>();// can engage only by user
    protected Permission permission = Permission.newInstance("xxx", "sweetie", "honey");
    protected DPermitter dPermitter = new DPermitter(permission);//TODO
    protected Hashtable<String, Integer> AlgDurations = new Hashtable<>();
    protected Fusion fusion = new Fusion(AlgDurations);
    http://fusion.getReqOverload() // an overload of requests on the brain
    http://fusion.getRepReq() // someone is negging and asking the same thing over and over again
    /*
    flight or fight skills may need access to the above fusion class booleans
    on the output methode of a skill this skills will load algorithms to the highest priority of the
noiron
    which carries algorithms :
    noiron.negativeAlgParts.add(Algorithm)
    * */
    public Personality(AbsDictionaryDB absDictionaryDB) {
        this.kokoro = new Kokoro(absDictionaryDB);
    }
    public Personality() {
        this.kokoro = new Kokoro(new AbsDictionaryDBShadow());
    }

    public ArrayList<AbsCmdReq> getdClassesLv1() {
        return dClassesLv1;
    }

    public ArrayList<AbsCmdReq> getdClassesLv2() {
        return dClassesLv2;
    }

    public ArrayList<AbsCmdReq> getdClassesLv3() {
        return dClassesLv3;
    }

    public Kokoro getKokoro() {
        return kokoro;
    }

    public Permission getPermission() {
        return permission;
    }

    public DPermitter getdPermitter() {
        return dPermitter;
    }

    public Hashtable<String, Integer> getAlgDurations() {
        return AlgDurations;
    }
```

```
    public Fusion getFusion() {
        return fusion;
    }
}
```

example subClass Personality1 :

Code:

```
package com.yotamarker.lgkotlin1;

public class Personality1 extends Personality{
    public Personality1(AbsDictionaryDB absDictionaryDB) {
        super(absDictionaryDB);
        // 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 DIBedTime(kokoro));
        // dClassesLv1.add(new DHungry());
        // dClassesLv1.add(new DIBurper(kokoro));
        dClassesLv1.add(dPermitter);
        dClassesLv1.add(new DIJoker(kokoro));
        dClassesLv1.add(new DIEliza(kokoro));
        dClassesLv1.add(new DILively(kokoro));
        dClassesLv1.add(new DIBurper(kokoro));
        dClassesLv1.add(new DIWeather(kokoro));
        dClassesLv1.add(new DICurrency(kokoro));
        dClassesLv1.add(new DIGamer(kokoro));
        dClassesLv1.add(new DSpeller());
        dClassesLv1.add(new DiSoulV3(kokoro));
        dClassesLv1.add(new DiPrefer(kokoro));
        dClassesLv1.add(new DIBukubukuchagama(kokoro));
        dClassesLv1.add(new DIAlerter(kokoro));
        dClassesLv1.add(new DiTglrAdapter(kokoro,"mommy",new TheSitterV2(kokoro, "mommy")));
        dClassesLv1.add(new DiMiniGamer(kokoro));
        dClassesLv1.add(new ThePet(kokoro));
        dClassesLv1.add(new DiMemoryGame(kokoro));
        dClassesLv1.add(new DiSaladSuggestor(kokoro));
        dClassesLv1.add(new DiB8Tri(kokoro));
        // dClassesLv1.add(new DCalculatorV1());
        http://dClassesLv2.add(new DSayer());
        dClassesLv2.add(new DiSayer(kokoro));
        // dClassesLv3.add(new DAlarmer());
        dClassesLv3.add(new DIDirty(kokoro));
        dClassesLv3.add(new DIHomer(kokoro));
        dClassesLv3.add(new DILifeFueler(kokoro));
        dClassesLv3.add(new DiTglrSkill(kokoro, "mommy"));
    }

    public Personality1() {
        super();
        dClassesLv1.add(new Detective(fusion));
        // dClassesLv1.add(new DJirachi());
        // dClassesLv1.add(new DIAutomatic(kokoro, master));
        // dClassesLv1.add(new DIBedTime(kokoro));
        // dClassesLv1.add(new DHungry());
        // dClassesLv1.add(new DIBurper(kokoro));
        dClassesLv1.add(dPermitter);
        dClassesLv1.add(new DIJoker(kokoro));
        dClassesLv1.add(new DSpeller());
        dClassesLv1.add(new DISoulV2(kokoro));
        // dClassesLv1.add(new DCalculatorV1());
        dClassesLv2.add(new DSayer());
        // dClassesLv3.add(new DAlarmer());
        dClassesLv3.add(new DIDirty(kokoro));
    }
}
```

ChobitV2 with the personality constructor :

Code:

```
package com.yotamarker.lgkotlin1;

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

public class ChobitV2 {
    protected String emot = ""; // emotion
    protected ArrayList<AbsCmdReq> dClassesLv1;
    protected ArrayList<AbsCmdReq> dClassesLv2;// can engage with friends and work related
    protected ArrayList<AbsCmdReq> dClassesLv3;// can engage only by user
    protected ArrayList<AbsCmdReq> dClassesAuto = new ArrayList<>();// automatically added and
engaged by time
    // algorithms fusion (polymarization)
    protected Hashtable<String, Integer> AlgDurations;
    protected Fusion fusion;
    // region essential DClasses
    protected Permission permission;
    protected DPermitter dPermitter;
    // endregion
    protected Neuron noiron;
    // sleep vars :
    protected Person activePerson = new Person();
    protected PrimoCera primoCera = new PrimoCera();
    // added :
    protected Kokoro kokoro; // soul
    protected Person master = new Person();
    protected String lastOutput = "";
    // standBy phase 260320
    protected TimeGate timeGate = new TimeGate();
    public ChobitV2(Personality personality) {
        super();
        this.AlgDurations=personality.getAlgDurations();
        this.fusion=personality.getFusion();
        this.kokoro = personality.getKokoro();
        permission=personality.getPermission();
        dPermitter=personality.getdPermitter();
        noiron = new Neuron();
        dClassesLv1=personality.getdClassesLv1();
        dClassesLv2=personality.getdClassesLv2();
        dClassesLv3=personality.getdClassesLv3();
        formAutoClassesList();
    }
public void loadPersonality(Personality personality){
    this.AlgDurations=personality.getAlgDurations();
    this.fusion=personality.getFusion();
    this.kokoro = personality.getKokoro();
    permission=personality.getPermission();
    dPermitter=personality.getdPermitter();
    noiron = new Neuron();
    dClassesLv1=personality.getdClassesLv1();
    dClassesLv2=personality.getdClassesLv2();
    dClassesLv3=personality.getdClassesLv3();
    dClassesAuto = new ArrayList<>();
    formAutoClassesList();
}
    protected void formAutoClassesList() {
        // adds automatic skills so they can be engaged by time
        for (AbsCmdReq dCls : dClassesLv2) {
            if (dCls.auto()) {
                dClassesAuto.add(dCls);
            }
        }
```

```
        for (AbsCmdReq dCls : dClassesLv3) {
            if (dCls.auto()) {
                dClassesAuto.add(dCls);
            }
        }
    }
    public String doIt(String ear, String skin, String eye) {
        ear = translateIn(ear);
        for (AbsCmdReq dCls : dClassesAuto) {
            inOut(dCls, "", skin, 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 translateOut(fusion.act(ear, skin, eye));
    }
    public String getSoulEmotion(){return kokoro.getEmot();}
    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 void inOut(AbsCmdReq dClass, String ear, String skin, String eye) {
        dClass.input(ear, skin, eye); // new
        dClass.output(noiron);
    }

    protected String translateIn(String earIn) {
        // makes sure the chobit doesn't feedback on her own output
```

```
        if (earIn.equals(lastOutput)) {
            return "";
        }
        return earIn;
    }

    protected String translateOut(String outResult) {
        // save last output served
        if (!outResult.isEmpty()) {
            lastOutput = outResult;
            this.timeGate.close();
            this.kokoro.standBy = false;
        }
        // standBy :
        else {
            if (!this.timeGate.isClosed()) {
                this.kokoro.standBy = true;
                this.timeGate.close();
            } else {
                this.kokoro.standBy = false;
            }
        }
        return outResult;
    }
}
```

DAISY CHAINING CHOBITS, AI PHARMACEUTICALS, AND CERABELLUMS

ChobitV2 with the personality constructor :

Code:

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

public class ChobitV2 implements thinkable{
    protected String emot = ""; // emotion
    protected ArrayList<AbsCmdReq> dClassesLv1;
    protected ArrayList<AbsCmdReq> dClassesLv2;// can engage with friends and work related
    protected ArrayList<AbsCmdReq> dClassesLv3;// can engage only by user
    protected ArrayList<AbsCmdReq> dClassesAuto = new ArrayList<>();// automatically added and
engaged by time
    // algorithms fusion (polymarization)
    protected Hashtable<String, Integer> AlgDurations;
    protected Fusion fusion;
    // region essential DClasses
    protected Permission permission;
    protected DPermitter dPermitter;
    // endregion
    protected Neuron noiron;
    // sleep vars :
    protected Person activePerson = new Person();
    protected PrimoCera primoCera = new PrimoCera();
    // added :
    protected Kokoro kokoro; // soul
    protected Person master = new Person();
```

```
    protected String lastOutput = "";
    // standBy phase 260320
    protected TimeGate timeGate = new TimeGate();
    public ChobitV2(Personality personality) {
        super();
        this.AlgDurations=personality.getAlgDurations();
        this.fusion=personality.getFusion();
        this.kokoro = personality.getKokoro();
        permission=personality.getPermission();
        dPermitter=personality.getdPermitter();
        noiron = new Neuron();
        dClassesLv1=personality.getdClassesLv1();
        dClassesLv2=personality.getdClassesLv2();
        dClassesLv3=personality.getdClassesLv3();
        formAutoClassesList();
    }
public void loadPersonality(Personality personality){
    this.AlgDurations=personality.getAlgDurations();
    this.fusion=personality.getFusion();
    this.kokoro = personality.getKokoro();
    permission=personality.getPermission();
    dPermitter=personality.getdPermitter();
    noiron = new Neuron();
    dClassesLv1=personality.getdClassesLv1();
    dClassesLv2=personality.getdClassesLv2();
    dClassesLv3=personality.getdClassesLv3();
    dClassesAuto = new ArrayList<>();
    formAutoClassesList();
}
    protected void formAutoClassesList() {
        // adds automatic skills so they can be engaged by time
        for (AbsCmdReq dCls : dClassesLv2) {
            if (dCls.auto()) {
                dClassesAuto.add(dCls);
            }
        }
        for (AbsCmdReq dCls : dClassesLv3) {
            if (dCls.auto()) {
                dClassesAuto.add(dCls);
            }
        }
    }
    public String doIt(String ear, String skin, String eye) {
        ear = translateIn(ear);
        for (AbsCmdReq dCls : dClassesAuto) {
            inOut(dCls, "", skin, 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 translateOut(fusion.act(ear, skin, eye));
    }
    public String getSoulEmotion(){return kokoro.getEmot();}
    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 void inOut(AbsCmdReq dClass, String ear, String skin, String eye) {
        dClass.input(ear, skin, eye); // new
        dClass.output(noiron);
    }

    protected String translateIn(String earIn) {
        // makes sure the chobit doesn't feedback on her own output
        if (earIn.equals(lastOutput)) {
            return "";
        }
        return earIn;
    }

    protected String translateOut(String outResult) {
        // save last output served
        if (!outResult.isEmpty()) {
            lastOutput = outResult;
            this.timeGate.close();
            this.kokoro.standBy = false;
        }
        // standBy :
        else {
            if (!this.timeGate.isClosed()) {
                this.kokoro.standBy = true;
                this.timeGate.close();
            } else {
                this.kokoro.standBy = false;
            }
        }
        return outResult;
    }

    @Override
    public String think(String ear, String skin, String eye) {
        return doIt(ear,skin,eye);
    }
}
```

so at this point you wonder why does the ChobitV2 class implements thinkable ?

this is a beef up, which enables daisy chaining chobits using a Brain class.
said brain class has input flow through a list of chobits(AGIs), and ending up in
a cerabellum class which is responsible for executing codes in the real world, meaning
speaking words, moving robotics, changing screen backgrounds, sending SMS and other such codes.

this daisy chain process enables AI pharmaceuticals, as can be seen in the Brain class documentation :
/*
* chobit hierarchy and responsibilities within the Brain class, which is a
* Chobit daisy chain.
*
* higher lv chobits : reality distortion on drug intake, and input
* translations(such as languages) lower lv chobits : drug addictions, context
* sensitive translation of outputs, and primitive behaviors. primitive
* behaviors : algorithm,s that do not require items and are not dependant on
* time or place
*/

to use the Chobit class you can remove the implements thinkable. however to use the Brian class add :

Code:
```
public interface actionable {
    //an interface for cerabellums, see CerabellumV2 for example
    public void act(String thought);
}
```

thinkable:

Code:
```
public interface thinkable {
    // an interface for the doIt function, see ChobitV2 for example implements
    public String think(String ear, String skin, String eye);
}
```

CerabellumV2 an example cerabellum class for the brain class to run IRL actions:

Code:
```
import android.app.Activity;
import android.content.Context;

public class CerabellumV2 implements actionable{
    private Boolean screen = true;
    private String time = "";
    private String place = "";
    private String item = "";
    private String person = "";
    private int number = 0;
    private MainActivity context=null;
    private RegexUtil regexUtil = new RegexUtil();
    public CerabellumV2(MainActivity context) {
        this.context = context;
    }
    @Override
    public void act(String thought) {
        /*
        * gets the chobit result string (chobit.doIt)
        * and converts it to an action*/
        if(thought.isEmpty()){return;}
        String action = regexCmd(thought);
        //case, functions or variables from the mainactivity are engaged as needed
        //modify as needed
        switch(action) {
            case "change screen":
                if(screen){context.screenFlip1();}
                else {context.screenFlip2();}
                screen = !screen;
                context.clearTxtBox();
```

```
                break;
            default:
                context.cerabellumSpeak(action);
        }

    }
    private String regexCmd(String thought){
        //populate the global vars and set the action (return string value to run in the act
functions switch case)
        //this is used for actions that use regexed params such as sending an SMS
        return thought;
    }
    private void clearGlobalVars(){}
}
```

and finally the brain class :

Code:

```
import java.util.ArrayList;

public class Brain {
    private ArrayList<thinkable> chobits = new ArrayList<>();
    private actionable action;

    /*
    * chobit hierarchy and responsibilities within the Brain class, which is a
    * Chobit daisy chain.
    *
    * higher lv chobits : reality distortion on drug intake, and input
    * translations(such as languages) lower lv chobits : drug addictions, context
    * sensitive translation of outputs, and primitive behaviors. primitive
    * behaviors : algorithm,s that do not require items and are not dependant on
    * time or place
    */
    public Brain(actionable action, thinkable...chobits) {
        super();
        this.action = action;
        for (thinkable chobit : chobits) {
            this.chobits.add(chobit);
        }
    }

    public void doIt(String ear, String skin, String eye) {
        String result = ear;
        for (thinkable chobit : chobits) {
            if (result.contains("#skin")) {
                result = result.replace("#skin", "");
                result = chobit.think(ear, result, eye);
                continue;
            }
            if (result.contains("#eye")) {
                result = result.replace("#eye", "");
                result = chobit.think(ear, skin, result);
                continue;
            }
            if (result.isEmpty()) {
                result = chobit.think(ear, skin, eye);
            } else {
                result = chobit.think(result, skin, eye);
            }
        }
        action.act(result);
    }
}
```

in the main activity :
declaring the globla variables :
var chii:ChobitV2? = null
var cerabellumV2:actionable?=null //TODO

var brain:Brain?=null

initializing variables (within onCreate function of the MainActivity):
chii = ChobitV2(Personality1(SharedPrefDB(this))) //builds a chobit with
chii!!.loadPersonality(Personality1(SharedPrefDB(this)))//reload a new personality if needed
cerabellumV2=CerabellumV2(this)//builds a class to run actuall real world actions
brain = Brain(cerabellumV2!!,chii!!) //builds a chobit daisy chain

example engaging the brain class :
brain!!.doIt(editText.text.toString(),"","")

DISKILLV2 :

A FASTER CLEANER VERSION OF DISKILL

```
public class DiSkillV2 extends AbsCmdReq {
    protected Kokoro kokoro; // consciousness, shallow ref class to enable
interskill communications
    protected DISkillUtils diSkillUtils = new DISkillUtils();
    protected Algorithm outAlg = null; // skills output

    public DiSkillV2(Kokoro kokoro) {
        super();
        this.kokoro = kokoro;
    }

    @Override
    public void input(String ear, String skin, String eye) {
    }
    @Override
    public final void output(Neuron noiron) {
        if (outAlg != null) {
            noiron.algParts.add(outAlg);
            outAlg = null;
        }
    }
}
```