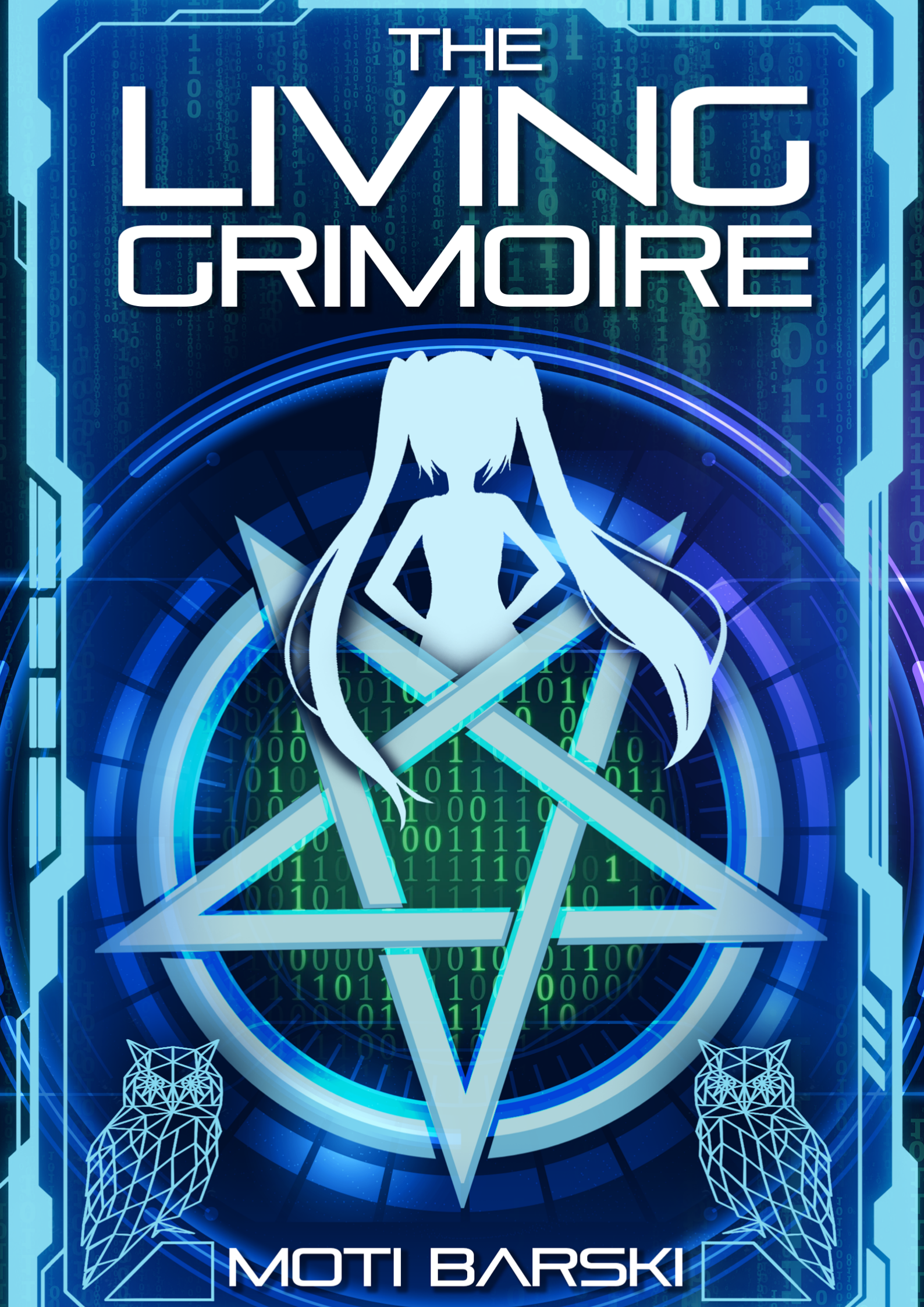
THE LIVING GRIMOIRE



MOTI BARSKI

intro

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 written approved by me personally.
you can use this for research.

name of the software : living grimoire : LG for short
what it is : an Artificial General Intelligence Software Design Pattern.

the LivinGrimoire enables adding skills, which are capabilities, by using one line of code(per skill).

benefits of using the livingrimoire design pattern

1. can prioritize skills against one another, pause and resume skills according to their priority
2. can que algorithm while running other algorithms
3. can engage several skills at once, and engage the right skills.
4. inter skill communication: skills can communicate with each other, pass data, and effect each other
5. interface is not used, which means it is applicable for all OOP programming languages, meaning cross platform porting.
6. has lots of auxiliary classes specialized for learnability and trigger management as well as misc. classes for time savings on common coding actions.
7. can form multistep algorithms, as well as abort algorithms while they run.

links :

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

the living grimoire forum :
https://www.yotamarker.com/f2-the-livingrimoire

getting started

https://github.com/yotamarker/public-livinGrimoire/tree/master/livingrimoire%20start%20here

to use the living grimoire choose the programming language you prefer.
there are 4 packages (directories). to keep LG projects neat and tidy :

1 LGCore : short for living grimoire core,
this are the core class files that compose the AGI software design pattern.

2 SkillsPkg : this directory should contain :

skill classes (naming convention : class names starts with Di)

AlgParts (naming convention : class names starts with AP)

3 HardwarePkg : hardware related classes or files not including the MainActivity class
for example : gps, vision processing, PID, Arduino, accelerometer, custom text to speech

hello world

```
Chobits chi = new Chobits();
chi.addSkill(new DiHelloWorld());
System.out.println(chi.think("hello","",""));
System.out.println(chi.think("","",""));
```

DiHello world is an example skill that says hello world as a reply to hello :

```
public class DiHelloWorld extends Skill{
    // hello world skill for testing purposes
    public DiHelloWorld() {
        super();
    }

    @Override
    public void input(String ear, String skin, String eye) {
        switch (ear){
            case "hello":
                super.setVerbatimAlg(4,"hello world"); // 1->5 1 is the highest
algorithm priority
                break;
        }
    }
}
```

so as you can see adding features to the chobit only takes :

1. adding the classes to the skill package
2. adding one line of code.

the Brain class

```
package LivinGrimoire;
/*********
 *intro *
 ********

 up until now, the LivinGrimoire was on par with the matrix learn scene.
 one line of code to add one skill.

 that is great, that is sci-fi turned real, that is the most significant coding
achievement in the history of time.

 but hey why stop there? why only be on par with the matrix and the human
brain ?
 what is beyond the matrix level? you already know

 cyberpunk>the matrix.
 one line of code to add a skill, but ALSO! 1 line of code to add a hardware
capability.

 ***********
 *Atributes*
 ***********

 the logicChobit is a Chobits attribute with logic skills. these skills have
algorithmic logic,
 and thinking patterns.

 the hardwareChobit is a Chobit attribute with hardware skills. these skills
access the
 hardware capabilities of the machine.
 for example: output printing, sending mail, sending SMS, making a phone call,
taking
 a photo, accessing GPIO pins, opening a program, fetching the weather and so
on.

 ********************
 *special attributes*
 ********************

 in some cases the hardware chobit may want to send a message to the logic
chobit,
 for example to give feedback on hardware components. this is handled by the
bodyInfo
 String.

 the emot attribute is the chobit's current emotion.
```

```
 the logicChobitOutput is the chobit's last output.

 **********************
 *hardware skill types*
 **********************

 assembly style: these skills are triggered by strings with certain wild card
characters
 for example: #open browser

 funnel: these are triggered by strings without wild cards.
 for example: "hello world"->prints hello world

 *************
 *example use*
 *************
 DiSysOut is an example of a hardware skill

 see Brain main for example use of the cyberpunk Software Design Pattern
 */
public class Brain {
    public Chobits logicChobit;
    public Chobits hardwareChobit;
    private String emot = "";
    private String bodyInfo = "";
    private String logicChobitOutput = "";
    public Brain() {
        logicChobit = new Chobits();
        hardwareChobit = new Chobits();
    }
    public void doIt(String ear, String skin, String eye) {
        if (!bodyInfo.isEmpty()){
            logicChobitOutput = logicChobit.think(ear,bodyInfo,eye);
            emot = logicChobit.getSoulEmotion();
        }
        else{
            logicChobitOutput = logicChobit.think(ear,skin,eye);
            emot = logicChobit.getSoulEmotion();
        }
        bodyInfo = hardwareChobit.think(logicChobitOutput,skin,eye);
    }
}
```

example use in main:

```
public class Main {
    public static void main(String[] args) {
        Brain b1 = new Brain();
        b1.logicChobit.addSkill(new DiHelloWorld());
        b1.hardwareChobit.addSkill(new DiSysOut()); // this skill prints output
        b1.doIt("hello","","");
        b1.doIt("","","");
        b1.hardwareChobit.think("test","", "");
    }
}
```

output:
hello world
test

```
import LivinGrimoire.Skill;

public class DiSysOut extends Skill {
    @Override
    public void input(String ear, String skin, String eye) {
        if (!ear.isEmpty() & !ear.contains("#")){
            System.out.println(ear);
        }
    }
}
```

method of operation overview

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

a skill sends out Algorithms if triggered.
said Algorithm is built with a list of parts (Mutatable class).
as the algorithm is running the action method of each mutatable is engaged outputting a string.

a running algorithm can also be aborted at any time by setting the mutatable algKillSwitch attribute to true.

see AP classes for Mutatable class examples

see DiHello world class for skill example

when an AP is completed, it's isActive attribute is set to false.

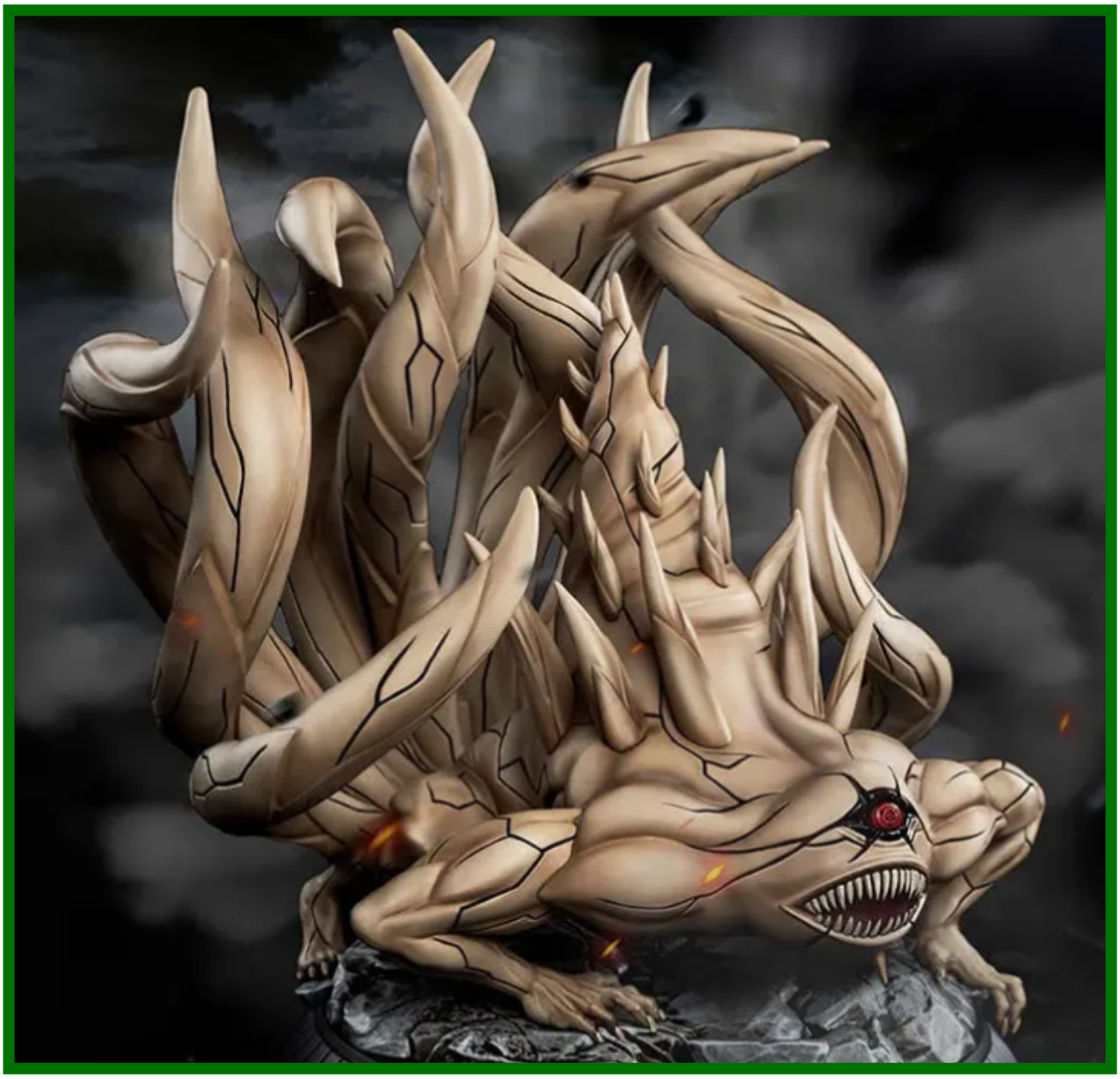
Kokoro class

```
import java.util.Hashtable;

/* this class enables:
communication between skills
utilization of a database for skills
in skill monitoring of which Mutatable was last run by the AI (consciousness)
this class is a built-in attribute in skill objects.
 * */
public class Kokoro {
    private String emot = "";

    public String getEmot() {
        return emot;
    }

    public void setEmot(String emot) {
        this.emot = emot;
    }
    public GrimoireMemento grimoireMemento;
    public Hashtable<String, String> toHeart = new Hashtable<>();
    public Kokoro(AbsDictionaryDB absDictionaryDB) {
        super();
        this.grimoireMemento = new GrimoireMemento(absDictionaryDB);
    }
}
```



AGI Juubi (10) main skill categories

essential skills for AI:

skills an AGI needs mostly :

1 Hungry : for eating or charging , sleep
2 wish-granter : 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
, recognize and avoid boredom

Chobits class

```
void setDataBase(AbsDictionaryDB absDictionaryDB)
```

```
Chobits addSkill(Skill skill)
```

```
void clearSkills()
```

```
void addSkills(Skill... skills)
```

```
public void removeSkill(Skill skill)
```

```
public Boolean containsSkill(Skill skill)
```

```
String think(String ear, String skin, String eye)
```

```
public String getSoulEmotion() {
    // get the last active AlgPart name
    // the AP is an action, and it also represents
    // an emotion
```

```
Kokoro getKokoro() {
    // several chobits can use the same soul
    // this enables telepathic communications
    // between chobits in the same project
```

```
void setKokoro(Kokoro kokoro) {
    // use this for telepathic communication between different chobits objects
```

```
Fusion getFusion()
```

livingrimoire core classes list

1. AbsDictionaryDB
2. Mutatable
3. APVerbatim:Mutatable
4. GrimoireMemento
5. Algorithm
6. Kokoro
7. Neuron
8. Skill
9. DiHelloWorld:Skill // logical skill for testing
10. Cerabellum
11. Fusion
12. Chobits
13. Brain
14. DiSysOut:Skill // hardware skill for testing

AI pharmaceuticals

chobits can be linked (chain effect) together
one chobits output can be used as input for the next chobit

thus AI drugs are programmable
when the "drugs" are input into the 1st chobit
it may engage for example reality morphing for input
which in turn, after passing through a certain skill in the
1st chobit will produce a different output before its sent to the main chobit.

drug/alcohol effects can be coded

as well as input to main language translation

suggested naming convention

AP : AlgParts class names should start with AP. APSay for example

***

Di : Skill (LivinGrimoire skill class names) should start with Di.
for example DiHelloWorld.

***

AX : auxiliary modules. these classes names can start with AX
for example AXLearnability

***

AXU: Auxiliary Modules for Upper Chobits, which are Chobit objects that process
input before it gets to a thinking chobit object.
these may include: AI pharmaceuticals, reality morph, input filtering (reality/pain),
data classification, and even AI hormones.

AXL: Auxiliary Modules for Lower Chobits, which are Chobit objects that process
input from thinking Chobits.
these may include translations, reply decorations, decoding, and custom dialects.

suggested skill summery convention format

suggested skill summery convention format

1 skill name
2 skill creator
3 skill description
4 skill triggers
5 notes (optional)

livingrimoire core utils

RegexUtil
this class eases regular expression utilization
and can also be used as a standalone class

DiSkillUtils
this class is an attribule of the Skill
this class has methods to ease generating common algorithms such as speaking output

for more refer to the DiHello world class (usage example)
or the livingrimoire UML wiki

Fusion class API

via a Chobits object you can get a reference to it's Fusion object
next you can add the reference to a skill via the skill's c'tor for example

the fusion class has an interesting methods

getEmot(): returns the last run alg part (which also represents emotion)

I believe a skill with access to the above methods, and thus overlooking the think
process summary is in essence **awareness**

if such skills also monitor body part states they ca be considered **self awareness**

parallelisms between the LivinGrimoire and the brain

sentience is an AIs ability to learn
and it is managed via the auxiliary modules

algorithm parts represent **emotions** and can be viewed via any skill with an added shallow reference to the Chobits objects Fusion
attribute, via the Fusions emot Attribute, which is the algParts class name.
a skill monitoring the active emotions is **awareness**.

self awareness would relate to the robots body parts which is not the same as
awareness.

programming language cheat sheets

coding cheat sheets for the java, python, swift, and the Koltin programming languages.

said sheets make programming easier and faster, as they concentrate commonly used codes.

the cheat sheets can be found in the "livingrimoire start here" directory in each respective language directory.

additionally, a keynote cheat sheet was added to the extras directory for coders who want to create presentations about livingrimoire skills, alg parts, auxiliary modules and such

LivinGrimoire directory guide

livingrimoire start here dir:
here you can find 4 versions of the livingrimoire (java, Koltin, python, swift)
choose according to the programming language you are using.

the extras dir has livingrimoire wiki txt files

Auxiliary modules

these class package(directory) contains classes that help writing AGI skills.

please refer to the complimentary book:
Auxiliary Modules for the LivinGrimoire

more algorithm building methods

the skill super class has methods to ease building algorithms.
you already saw an example in the DiHelloWorld skill,
here are some more :

```
public class Skill {
    protected Kokoro kokoro = new Kokoro(new AbsDictionaryDB()); //
consciousness, shallow ref class to enable interskill communications
    protected DISkillUtils diSkillUtils = new DISkillUtils();
    protected Algorithm outAlg = null; // skills output
    protected int outpAlgPriority = -1; // defcon 1->5
```

```
    public Skill() {
        super();
    }
    // skill triggers and algorithmic logic
    public void input(String ear, String skin, String eye) {
    }
    // extraction of skill algorithm to run (if there is one)
    public void output(Neuron noiron) {
        if (outAlg != null) {
            noiron.insertAlg(this.outpAlgPriority,outAlg);
            outpAlgPriority = -1;
            outAlg = null;
        }
    }
    public void setKokoro(Kokoro kokoro) {
        // use this for telepathic communication between different chobits
objects
        this.kokoro = kokoro;
    }
    // in skill algorithm building shortcut methods:
    protected void setVerbatimAlg(int priority, String... sayThis){
        // build a simple output algorithm to speak string by string per think
cycle
        // uses varargs param
        this.outAlg = this.diSkillUtils.simpleVerbatimAlgorithm(sayThis);
        this.outpAlgPriority = priority; // 1->5 1 is the highest algorithm
priority
    }
    protected void setVerbatimAlgFromList(int priority, ArrayList<String>
sayThis){
        // build a simple output algorithm to speak string by string per think
cycle
        // uses list param
```

```
        this.outAlg = this.diSkillUtils.algBuilder(new APVerbatim(sayThis));
        this.outpAlgPriority = priority; // 1->5 1 is the highest algorithm
priority
    }
    protected void algPartsFusion(int priority,Mutatable... algParts){
        // build a custom algorithm out of a chain of algorithm parts(actions)
        this.outAlg = this.diSkillUtils.algBuilder(algParts);
        this.outpAlgPriority = priority; // 1->5 1 is the highest algorithm
priority
    }
}
```

A DETAILED ELABORATION OF THE MOST EFFICIENT WAY TO PROGRAM.
LEARN ABOUT THE ARTIFICIAL GENERAL INTELLIGENCE SOFTWARE DESIGN PATTERN CALLED THE LIVING GRIMOIRE, WHICH AMONG ITS MANY CAPABILITIES, ENABLES ADDING SKILLS USING ONLY ONE LINE OF CODE PER SKILL.