



MPL

Music Processing Language

Project Manager: Bo Wang

Language Guru: Yilin Xiong

System Architect: Shengyi Lin

System Integrator: Ying Tan

Verification and Validation: Mengting Wu

Why MPL?

Break the traditional way to compose note by note

Make the composition process programmable, efficient and interesting.

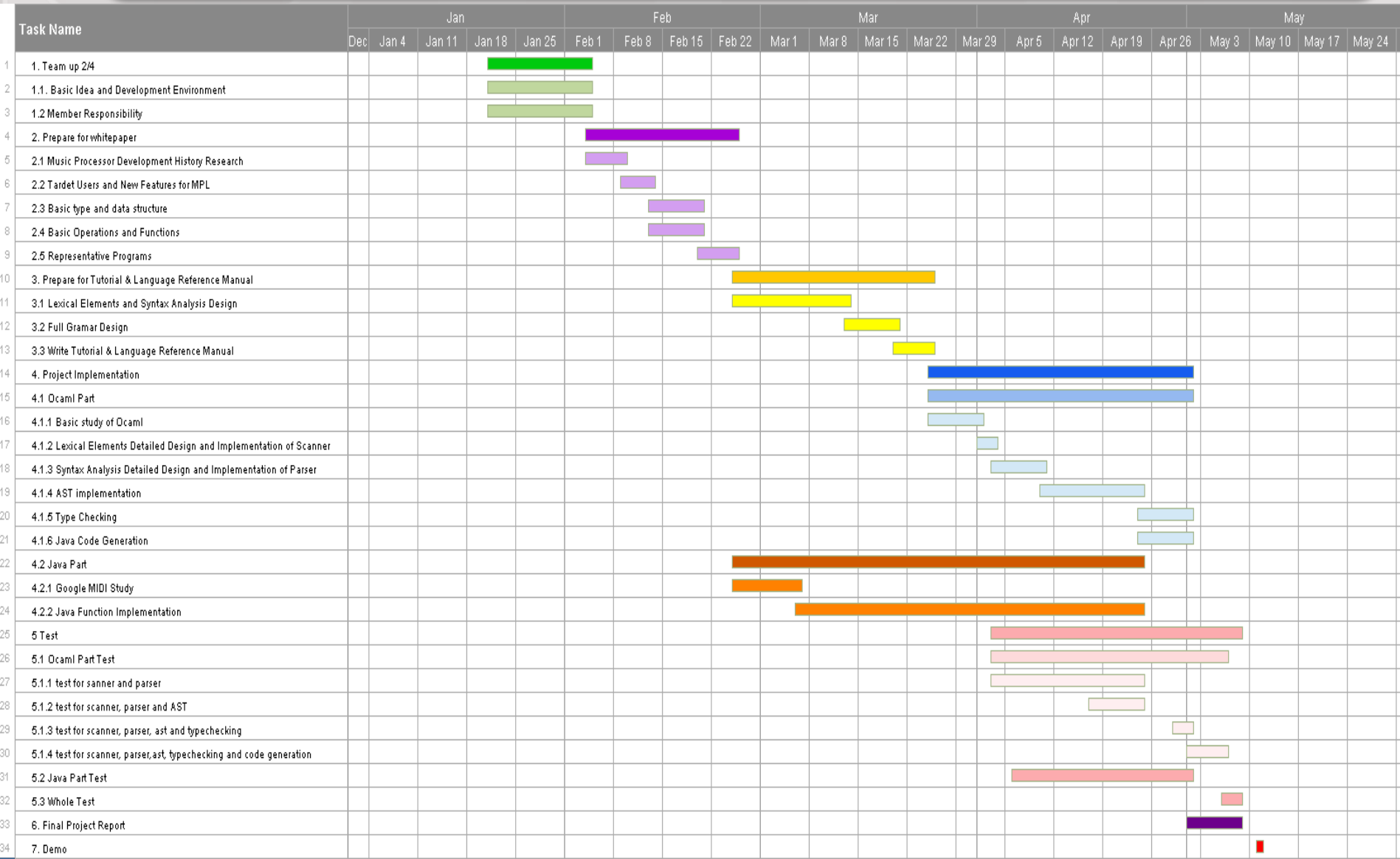
– Target Users

- Professional musicians and music amateurs
- People love both music and programming

– Sparking Features

- Simple data structure
- Convenient Operations
- Rich Functions

Time Plan



MPL in One Slide

```
Melody addChords(Melody melody) {  
    Melody newMelody = Melody();  
    for(int i = 0; i < melody.getLength(); i++) {  
        Note note = melody.getNote(i);  
        Note chord = note - 12;  
        newMelody.addNote(chord);  
    }  
    return newMelody;  
}
```

Variable Definition Use
Type Construction

Calculation of music

Built-in function for music type

```
void main(string arg[]) {  
    Music music = read("twinkle_twinkle0.mid");  
    Melody melody = music.getTrack(0).getMelody();  
    Melody newMelody = addChords(melody);  
    Track tracks[] = {music.getTrack(0), Track(newMelody, PIANO)};  
    Music newMusic = Music(tracks);  
    write(newMusic, "new_twinkle_twinkle.mid");  
    // Finished  
}
```

Built-in function for
general purpose

Call user defined
function

Function
Definition

For Loop

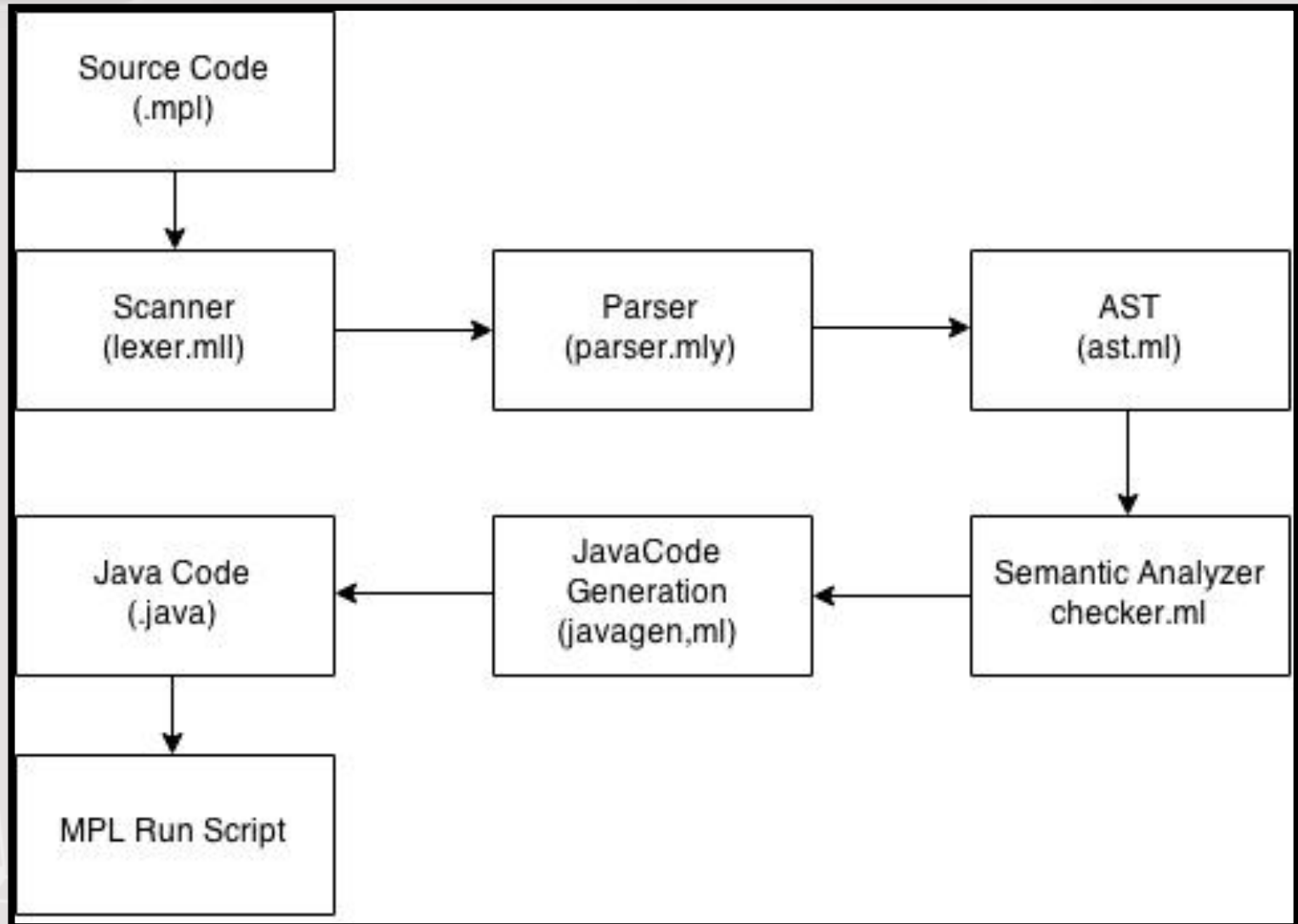
Array
Initialization

Comments



Key Features Demo

How it works



Role & Responsibility

| Module | Responsible Members |
|---------------------|-----------------------------------|
| Scanner | Mengting Wu, Bo Wang |
| Parser | Yilin Xiong |
| AST | Bo Wang, Mengting Wu |
| Checker | Yilin Xiong, Ying Tan |
| Code Generation | Yilin Xiong, Bo Wang, Mengting Wu |
| Java Implementation | Shengyi Lin, Ying Tan |
| Test | Shengyi Lin |

Symbol Table

- Environment Stack

Keep Scope: Number each blocks.

Push the number of the current block when go into the block, and pop out after go out.

- Symbol Table

Name each variable with name “scopeNumber_name” and store its type with name into HashMap.

Type is stored as string.

Function type: void_argType_returnType

Symbol Table

```
//Push: block 0    Stack: 0
void main(string arg[]) {
    // Push: block 1    Stack: 1 0
    Music music = read("twinkle_twinkle2.mid");
    int i;
    for(i = 0; i < music.getNumberOfTracks(); i++) {
        // Push: block 2    Stack: 2 1 0
        Track track = music.getTrack(i);
        track.setTimbre(PIANO);
        // Pop: block 2    Stack: 1 0
    }
    write(music, "twinkle_twinkle3.mid");
    // Pop: block 1    Stack: 0
}
```

Symbol Table:

| Variable | Type |
|----------|----------------------------|
| l_arg | array_string |
| l_music | Music |
| l_i | int |
| 2_track | Track |
| main | void_array_s tring_void |

Run Time Environment

- Ocaml (Frontend for translator)
- JDK 1.6 (JVM)
- MPL.jar: (Java backend with midi API)

users can download from <https://github.com/PLT-MPL/MPL/tree/master/ShellScript> to generate midi file.

MPL.sh

Help user to test MPL code more conveniently.

Script and Test Plan

Command:

MPL.sh <Path/To/ShellScript/Folder> <Input/MPL/
Filename> <Output/Executable/Filename>

Test Plan:

| Modules | Test Result |
|-----------------------|-------------|
| Hello World | √ |
| Note Initialization | √ |
| Note Manipulation | √ |
| Melody Initialization | √ |
| Melody Manipulation | √ |
| Track Initialization | √ |
| Music Initialization | √ |
| Music Manipulation | √ |
| Music Creation | √ |
| Self-defined Function | √ |



DEMO

Lessons Learnt

- Flexible timeline are vital important.
- Start work before considering too much.
- Functional languages are sooo interesting.
- We should have taken Computer Science Theory class before it.



Thanks!