EasyPet
Language Reference Manual

Team 13:

Bo Liu	bl2400
Xiaolong Jiang	xj2127
Liang Wei	lw2425
Shen Wang	sw2613
Weikang Wan	ww2267

REVISION HISTORY				
Revision	**Version**	**Description of Change**	**Changed By**	**Effective Date**
1	0.1a		Team 13	03/22/11
2				
3				

Table of Contents

Tutorial

1. Introduction

The language we designed is named EasyPet. With this language a developer can design electronic pets conveniently. For example, an electronic dog can be designed with several actions for people to keep. They can not only feed the dog, but also train it so that it can grow up healthy.

The first part is an introduction of the language. In this part, our aim is to show the essential elements of the language. For example, the basic method is to design a panel. However, the details of the language won't be discussed. We aim to help developers understand the basics of this language to design simple program as quickly as possible.

Some sample codes will be explained in this part. Developers can learn from them to know how they can develop their first program with EasyPet. The basic aspects of the language EasyPet are data types, variables, arithmetic expressions, iteration, array, and functions.

2. Getting Started

The language is used to design electronic pets for children or adults to kill their spare time. Therefore, the first program is an easy one: The dog snoopy can bark to say "hello world" when users press the button. The sample code is as following:

```
Item Dog {
        String name;
        int age;
        int speed;
        arraylist<image> images;

        Dog(name, age, speed);
}
```

```
// another file

Panel {
	Dog dog();
	dog.name = "Snoopy";
	dog.age = 1;

	/*button function
	*When user click the button, change the image
	* while playing the sound
	*/
	def_btn Hello() {
		name = “Hello”;
		dog.images.add(“src/img_say_hello_world”);
	}
}
```

This is a sample code of EasyPet language. The program consists of functions and variables. The functions are used to specify the computing of the operation to be done and the variables are used to store the data which is needed in the process of computing.

Actually, there are three kinds of functions in this language. The first kind of function is defined by the keyword “def_timer”. This kind of function will be called by the timer. For example, in this sample, the age of the pet will increase with the argument speed when timer calls it every one time unit.

The second kind of function is defined by the keyword “def”. This kind of function will be called by other functions. For example, in this sample code, the function which is defined by “def” is called by the function ageGrow, which leads to the death of the pet when its age becomes 10.

The third kind of function is defined by the keyword “def_btn”. This kind of function can only be defined in the scope of Panel. This function plays a similar role with ActionListener in Java because it can design a button in the panel and make response to the operation to the user.

Item is similar the definition of class in Java. In this sample, an item whose name is Dog is defined. There are four properties in this item, including name, age and speed, images. And also, several functions are defined. Dog(name, age, speed, images) is a constructor function, which is used to initialize the object. Another two functions have already been introduced in the previous paragraph.

Panel is the most basic element in the language EasyPet. The functions to implement the functions of the buttons are defined in it. And also, the initialization of Item is defined in Panel. Moreover, it plays a similar role like the main() function in the C and Java language. All the functions should be called in Panel because in fact the function can only be called when an Item is initialized.

The basic method to implement the communication between functions is similar to the method in the C and Java language. The calling function needs to provide a list of values as arguments with the called function. The arguments need to be surrounded by the parentheses after the function name.

The statements of a function are enclosed by { }. In this sample, the operation to update the image for the pet snoopy is implemented in the { } of function Hello().

3. Variables and Arithmetic Expressions

This program uses the formula $T(n) = \begin{cases} T(n-1)+1 & n \leq 8 \\ 9 & n > 9 \end{cases}$ to simulate the growing process of the pet “snoopy”: while snoopy is younger than 8, it grows according to the formula T(n) = T(n-1)+1;when snoopy reaches 9, then his height stops growing.

```
Item Dog{
        String name;
        int age;
        int height;

        Dog(name, age, height);

        /*growing up function
        * The dog follows the formula to grow up
        *When he reaches 9 years old,he stops grow up
```

```
        */
        def_timer  void growUp(Item Dog )  {
                if( age<=8){
                                height = height +1;
                }
                else  {
                        height = 9;
                }
        }
}

Panel {
        Dog dog();
        dog.name = "Snoopy";
        dog.age = 1;
 }
```

This is a program designing a dog pet named snoopy. It is the same as the hello world program which consists of the default variables. The goal of the program is to implement the growing up function for the pet following the formula. In this program, we will introduce variables, comments, and conditional statement.

The four lines:

*/*growing up function*
** The dog follows the formula to grow up*
**When he reaches 9 years old, he stops to grow up*
**/*

They are comments which briefly describe the function to make developers understand the function easily. Any characters between /* and */ are ignored by the compiler.

In our language, variables are String name;, String ;int age, int age = 0;
float = 0;int height, int speed, int value. String name, int age, int height, int speed are variables defined in the initialization of the item Dog. These are the necessary attributes for every pet which should define in front. Int speed, int value are the variables for the function growUp() which is called by the timer. In certain time unit, it will get called. This makes it easier for users to develop. In language, the timer is fully defined so

it saves time for the developer, they don't have to hardcode the timer. Just add "_timer" behind the function to declare this function is involved with timer.

In our language, the conditional statement is the same as it is in java. For the growing up function, the program begins with the statements *value % 100 == 0 && value/100 <=8.* Every second, the status calls the function growUp(), until the value reaches 100. The height is growing at the formula until the age is more than 8.

4. "for" statement

This part is aiming to show the sample code of for statement. For this language, for can be used to implement a series of same operations. The sample code is as following:

```
Item Dog {
	String name;
	int age;
	int speed;
	arraylist<image> images;

	Dog(name, age, speed, images);
}

// another file

Panel {
	Dog dog();
	dog.name = "Snoopy";
	dog.age = 1;

	/*button function
	*When user click the button, change the image
	*/
	def_btn void  OffTraining() {
		int numOfSuccess = 0;

		for(int i = 0; i < 5; i = i +1) {
```

```
                float r = random();
                if(r > 0.5) {
                        numOfSuccess = numOfSuccess + 1;
                }
        }

        if(numOfSuccess > 3) {
                dog.images.add("src/img_train_successfully");
        } else {
                dog.images.add("src/img_train_failed");
        }
    }
}
```

The statements of the "for" block are executed for several times. The number of times is decided by the developer. For example, in this sample code, the number is 5. Then random() (which is provided by the library) will be called for 5 times to generate 5 random number between 0 and 1. If the number is bigger than 0.5, then the training is successful.

Actually, we also provide while as an alternative. Developers can choose the keyword they like to implement the same operations.

5. Symbolic Constants

In our language, we use #define as symbolic constants in case if the developer needs to change a constant which is hard to find in the numerous lines of code. In this way, it is easier to understand and implement a meaningful name. A #define line defines a symbolic name or symbolic constant to be a particular string of characters:

#define name replacement list

Thereafter, any occurrence of the name (not in quotes and not part of another name) will be replaced by the corresponding replacement text. The name has the same form as a variable name: a sequence of letters and digits that begins with a letter. The replacement text can be any sequence of characters; it is not limited to numbers.

```
#define  UPPERAGE  8  /* upper limit of age of growing up */
#define  UPPERHEIGHT 9  /* upper limit of height that the pet won't grow up*/

Item Dog{
        String name;
        int age;
        int height;
        Dog(name, age, height);

        /*growing up function
        * The dog follows the formula to grow up
        * When he reaches 9 years old, he stops grow up
        */

        def_timer  void growUp(Item Dog)   {\
                if(age <= UPPERAGE) {
                        height = height +1;
                }
                else  {
                        height = UPPERHEIGHT;
                }
        }
}

Panel {
        Dog dog();
        dog.name = "Snoopy";
        dog.age = 1;
 }
```

UPPERAGE and UPPERHEIGHT are symbolic constants which are defined in front of the code which is easier for users to find and understand. Symbolic constant names are conventionally written in upper case so they can be readily distinguished from lower case variable names. Notice that there is no semicolon at the end of a #define line.
Declarations

All variables must be declared before use. In our language, it can be declared in the Item. A declaration specifies a type, and contains a list of one or more variables of that type as

```
String name, int age, int height;
```

Variables can be distributed among declarations in any fashion; the lists above could well be written as

```
String name ;
int  age;
int  height;
```

The latter form takes more space, but it is more convenient for adding a comment to each declaration for subsequent modifications.

6. Arraylists

In our language we also provide developers with arraylist data type to store a series of data. “arraylist” is useful in the development and we will use a sample code to show how to use this type. The sample code is as following:

```
Item Dog {
        String name;
        int age;
        int hungry;
        arraylist<image> images;

        Dog(name, age, hungry, images);
}

// another file

Panel {
        Dog dog();
        dog.name = "Snoopy";
        dog.age = 1;
```

```
        dog.hungry = 10;

        /*button function
        *When user click the button, change the image
        */
        def_btn void  Feed() {
                if(dog.hungry > 0) {
                        dog.hungry = 0;
                        dog.images.add(“src/img_feed_pet”);
                }

                for(int i = 0; i < dog.images.length; i++) {
                        if(dog.images.get(i) == “src/img_feed_pet”) {
                                dog.images.remove(i);
                        }
                }
        }
}
```

This sample is used to feed the pet. If the hungry of the pet is bigger than 0, then the pet is able to eat something. So at first a conditional block is needed. The if/else block will judge whether the pet can eat or not. If the pet can eat, then the hungry will be set to 0 which means the pet is full and the image will be added into the queue for the GUI to show. Because the image will only be shown for once, so it will be removed from the queue when the pet finish eating.

Actually, the arraylist in the language supports many other types. In the sample code, the arraylist with type “image”. In our language, some other types, such as string, float and other data types are also supported.

7. Functions

In EasyPet, function is used to implement computing operation. Actually, it is usually used to activate different statuses of the pets and update the image or sound of them. The sample code is as following:

```
Item Dog {
        String name;
        int age;
        int happy;
        arraylist<image> images;

        Dog(name, age, happy, images);
}

// another file

Panel {
        Dog dog();
        dog.name = "Snoopy";
        dog.age = 1;
        dog.happy = 0;

        /*button function
        *When user click the button, change the image
        */
        def_btn void  PlayBall() {
                dog.images.add(“src/img_play_ball”);

                float r = random();
                if(r > 0.5) {
                        dog.happy = 100;
                }


                if(dog.happy == 100) {
                        dog.images.add(“src/img_play_happily”);
                } else {
                        dog.images.add(“src/img_play_unhappily”);
                }
        }
}
```

This function is used to train the pet. The level of training of the pet decreases as the time flies and this function can help the pet to keep healthy. When the pet lacks training, the function can be called by the master of the pet. If the training is successful, then the status will be transferred to that the training is sufficient.

For functions, there are different return types of them. The return values can be used in other computing operations. In the sample code, the return type is void, which means that there is no return value.

There are two patterns to define functions in EasyPet.
The first pattern is:

```
def return-type function-name (parameter declarations, if any) {
        declarations
        statements
}
```

This pattern is used just like the sample code.

The second pattern is:

```
def-btn function-name (parameter declarations, if any) {
        declarations
        statements
}
```

This pattern is similar to the action listener of button in Java. It is just used to respond to the operation of the user. So there is no return type for this pattern of function.

In this sample code, numOfSuccess is a declaration to store the number of successful training. Other codes are statements to the computing operation. The function will update the image of the pet according to the result of the computing.

8. Arguments – Call by Value

In our language, the arguments of the functions whose types are primitive are passed by value. It means that the called function will obtain the arguments in temporary variables

just like the C language, not the original variables. Therefore, the change of the argument will have no effect on the original variable. The sample code is as following:

```
Item Dog {
        String name;
        int age;
        int speed;
        int hungry;
        arraylist<image> images;

        Dog(name, age, speed, hungry);

        /* called by Timer */
        def_timer void becomeHungry(int speed) {
                hungry = hungry + speed;
                if(hungry == 100) {
                        dieNaturally();
                }
        }

        /*called by becomeHungry() */
        def void dieNaturally() {
                for (int i = 0; i < images.length; i++) {
                        images.remove(i);
                }
                images.add(“src/img_dead”);
        }
}

// another file

Panel {
        Dog dog();
        dog.name = "Snoopy";
        dog.age = 1;
}
```

In this sample code, we can see that there is an argument which is named “speed” in the function becomeHungry(). Parameter speed will determine the speed of the pet to become hungry. The parameter is called by this function. However, the original value of it cannot be changed by the function because the argument is called by value.

In this sample code, the hungry will become bigger and bigger as the time elapsed. If the hungry is 100, then another function dieNaturally() will be called which means that the pet is die from hungry. All the images for each status of the pet in the image arraylist will be removed. Instead of these images, another image which is death will be added into the arraylist and showed to the user.