Kodename

Detailed Design Document

Joe Ancona
Stephen Chung
Miracle Okubor
Benjamin Salah
Isaac Tyan

Abstract:
Kodename is an feature-rich application developed by the Kodename team that aims to combine and improve upon the best features of Karel and Scratch for the purposes of teaching children in a fun and interactive manner the fundamentals of programming. This includes learning some pillars of programming such as modularity, reusability,and other skills that can be easily applied for any other programming languages they may encounter in the future. Kodename attempts bridge the application gaps present in Karel and Scratch by both avoiding the need to understand deep language syntax while still creating a programming vocabulary bears resemblance to other mainstream languages. Kodename uses the Karel programming language to teach the user programming and with a user interface, Kodename aims to eliminate the need for the user to focus on syntax. For individual modules check API.

Design:
See attached java doc for appropriate modules and routines.

Pre and Post conditions, Error Handling, and Invariants(check appropriate routine in API):

Project Module:
public Project(World world, Karel karel, ArrayList<CustomCode> customCode, Controller controller)
public void setWorld(World world)
public void setKarel(Karel karel)
public void setCustomCode(ArrayList<CustomCode> customCode)
public void setController(Controller controller)

Karel Module:
public Karel(World w, int x, int y)
public void setPosition(int x, int y)
public void turnRight()
public void turnLeft()
public void move()
public void putDown()

World Module:
public World(int x, int y)
public Contents getContents(int x, int y)
public void setContents(int x, int y, Contents c)

LoopCode Module:
public LoopCode(String condition, ArrayList<Code> body)
public LoopCode(int n, ArrayList<Code> body)
public void setCondition(String c)

```
public void setBody(ArrayList<Code> b)
public void setCounter(int c)
```

Exports, Imports, and Subparts: check API or UML Diagram.

Test Cases:
All of our pre and post conditions have the potential to break the system, module, or routine if left unchecked.

Questions:

1. What are the most likely modifications that a client might ask for or improvements that you might want to provide? How easy would it be for you to make those changes? What modules would be affected, and how extensively? Would conflicting design considerations cause some modifications to be infeasible or to require extensive changes?

   A more versatile, robust method of creating a World e.g. a WYSIWYG visual editor for a World. This would be difficult and time-consuming, requiring a large amount of UI work (e.g. in Swing). World would be affected, plus a new module would be created to present the UI to the user.

2. What are the implications of alternate design decisions on higher/lower/same level modules?

   Minimal implications with respect to interoperability between modules i.e. integration of modules. If interfaces are well-defined and documented, then changes within a module should not require code changes in other modules. Performance optimizations could be implemented on a module-by-module basis.

3. Do you want to add anything to your design to make debugging and testing the system easier? This may be discussed separately or included in the discussion of some modules.

   Use restrictive access modifiers such that other modules have limited ability to peek inside or modify each other (e.g. make them black boxes). Create JUnit tests for each module before or as we code the modules.

4. Management Issues: What implementation strategies will you use? Who will be responsible for the further design and coding of the pieces? Estimate the time needed for the development of the major sections of your system based on the estimated size and complexity. Remember to concentrate on the critical pieces of the system.

   Implementation strategy: divide the work to minimize interdependence on team members i.e. have well-designed interfaces such that person A can develop code without waiting for person B to finish something. All members will be responsible for designing and coding their allocated modules. The time needed to develop the basic / essential level of our program will be three weeks. Critical pieces: UI, Karel interpreter, representation of world and robot.

5. Have all interfaces to secondary storage or hardware devices been discussed?

   Karel programs / sessions can be saved to disk and loaded at a later time.