



Teaching Kids Programming

Learn to Teach the TKP Java Courseware

By Lynn Langit



TKP Java Lesson Plans

Lynn Langit

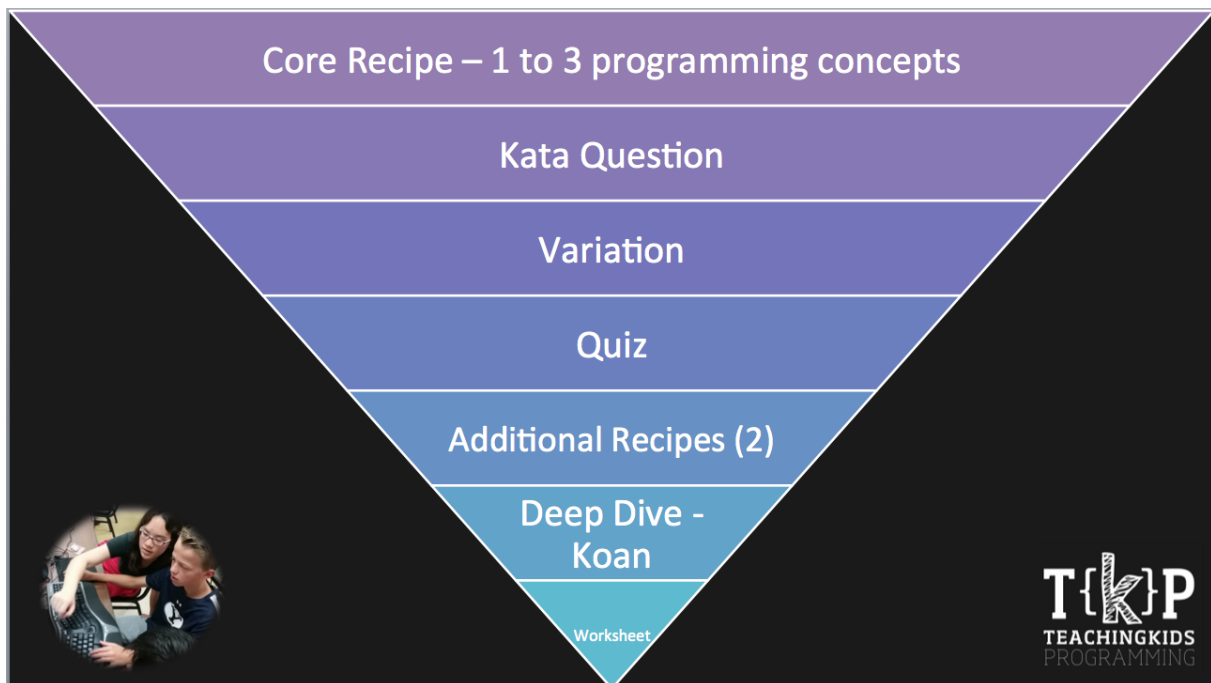
This project can be followed at:

<https://www.penflip.com/lynnlangit/tkp-lesson-plans>

©2017 Lynn Langit

Contents

1 TKP Instructional Design Information



Topics Covered:

- 5 Steps to teach TKP Courseware
- About The TKP Intentional Method
- What are the Parts of each TKP Course?
- How TKP uses Agile Technical Practices and Terms
- Why Pair Programming
- Why Group (Mob) Programming
- TKP Classroom Management Tips and TKP Virtual Proctor Utility
- Guide to TKP Java courseware and source code on Github
- Guide to Authoring TKP Recipes
- TKP courseware for other computer languages - SmallBasic, C#, T-SQL and more
- About TKPJava and the APJava Exam

5 Steps to teaching TKP Courseware

TKP Courseware is designed to be **instructor-led by K-12 teachers** (middle school is our core audience). TKP teachers do not need to be professional Java developers, but they should expect to learn programming by completing the TKPJava courseware themselves prior to teaching it. TKP Java Courseware is a set of custom Java (code) libraries, teacher lesson plans (Penflip) and YouTube screencast videos.

*Note: If you are familiar with coding in Java, you can start by directly downloading the **TKPJava** code and Eclipse editor settings from Github (detailed desktop installation instructions on Github) :octocat: [here](#).*

Listed below are the steps to prepare to teach TKP courseware:

1. **Read about TKP Instructional Design Methods.** The core information is on this page. There are links to more detailed and background reading on this page.
2. **Do the recipes / lessons in the course you plan to teach as a student.** This means coding the lessons in an editor or IDE yourself.
3. **Practice teaching to one or more students (kids or adults).** Particularly if you are new to programming, TKP teachers report practice teaching to a smaller group in advance of the 'main' class helping them to be more comfortable teaching.
4. **Review the available resources for each TKP course.** Resources include written explanation, mostly line-by-line for each course in Penflip, also recipe answer files. Additionally some recipes have answer videos on YouTube.
5. **Use TKP Utilities.** Our current utility is the TKP Virtual Proctor / website, which allows you to 'see' student's work from one browser window. More utilities are planned.

*

About The TKP Intentional Method

We at TKP use a new method of teaching children programming. We call this the **intentional method**. Our work consists of customized courseware and teaching techniques.

:red_circle: **CORE IDEA:** *The Intentional Method is teaching by guiding kids (working in pairs) to translate English comments (the intention) into runnable code. The principal programming language that we use is Java.*

We have created small experiments using the Intentional Method in other languages. These include Microsoft SmallBasic, T-SQL and for Microsoft Kodu (visual programming). We are also working on some IoT (Data) projects.

There are a few key concepts that we attempt to follow in writing and in teaching all our TKP courses. These concepts include the following:

- 1) **One line of English = One line of code** - exploration of API via Intellisense is a key aspect here. We've paid careful attention to 'leveling' using both our TKPJava API to 'wrap complex

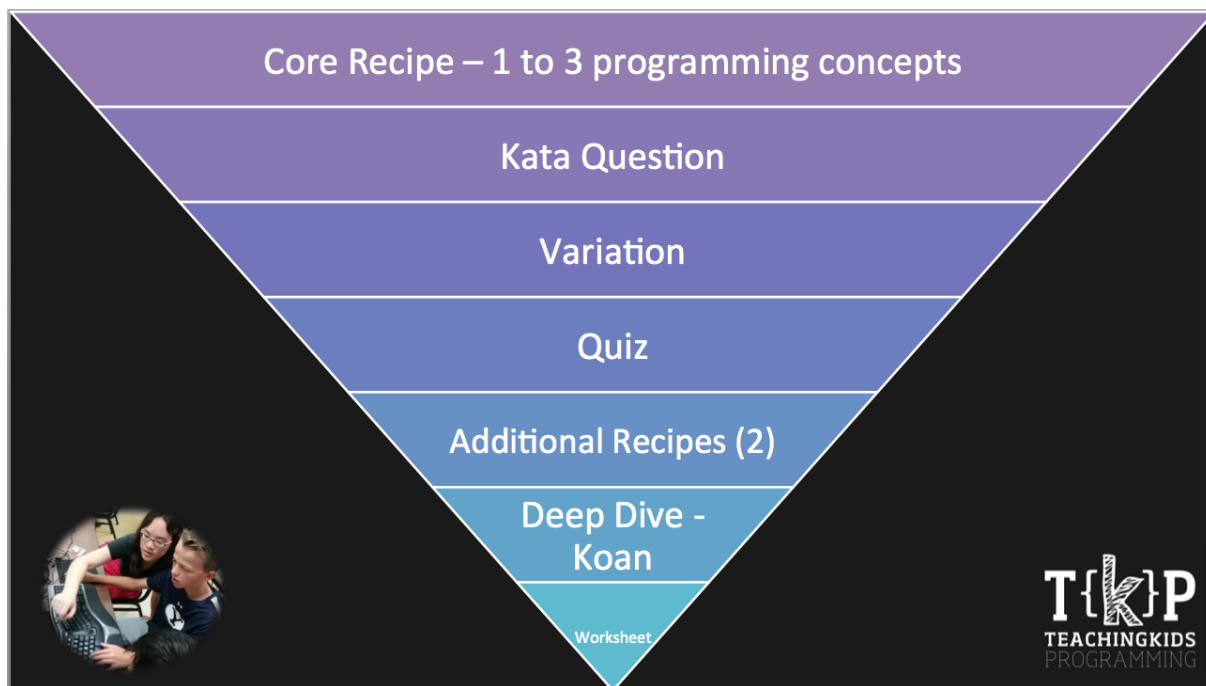
core Java concepts' and also creating a set of standard English comments to represent common Java concepts, for example in 'Set the value of the current length to 40', we are deliberately using 'current length' to indicate that the student should first search via Intellisense to see if a variable named 'length' is already in scope, and if not, should create a new variable of that name.

2) **Code one+ line(s) and then run your program to verify** that your translation was correct (lessons are meant to be completed in a specific order (and have line numbers at the end of each line to indicate that order - multi-part lines are numbered 1.1, 1.2, etc..

3) After verification, delete that line of English comments. Our lessons and API are meant to be discoverable by the students (with direction by the teacher). The teachers should encourage the students to use Intellisense (Ctrl+Space, or Cmd+Space) and to read the drop-down list of available items to see which one matches the English. Teachers should **NOT** type the Java code before the students do so. Teachers should **NOT** have the students copy the Java code from the board, rather they should ask the students to read the English comments aloud if students are stuck. Teachers should also direct students to read/use examples in the Javadoc. **Teaching API exploration and use of tools is a key aspect of teaching TKPJava.**

:pushpin: Note: For Teacher Preparation only, we have added the 'answers' to all course lessons to the source GitHub Repository at this [location](#)

4) **Concept mastery via multiple methods** of use - there are 6-8 recipes (lessons) per course which cover the particular programming concepts being taught in deliberately duplicated and increasingly complex ways. We design courseware that is carefully leveled and paced for students to learn and master 1 or 2 new computational concepts at a time. Below is a graphic which shows the parts (lesson types) in each TKPJava course in suggested order of teaching, i.e. the top level item (recipe) is designed to be first lesson taught in a course, etc....



Here is a 20-minute welcome video for K-12 teachers on how to use TKPJava courseware resources.



What are the Parts of each TKP Course?

1) **Recipe** – This is a **teacher-led** coding lesson (recipe) that teaches 1-3 programming concepts, such as ‘what is a class or a method?’ Intended for students to pair program as directed by the teacher, translating one line of English into one line of code and then running the program to verify the correct translation after each line is translated.

Recipes are written to be translated and run in a SPECIFIC order. This order is indicated by the line numbers at the end of each line of English comments. The reasons for the ordering are as follows:

- Introduce concepts one by one (sometimes directly reinforcing with repetition)
- Slow movement away from one-for-one English comment (meta) language to TKPJava library (term) language. In other words, we start with direct matches, i.e. ‘Show the Tortoise’ becomes ‘Tortoise.show();’ and move toward less direct translation, i.e. ‘Make the Tortoise move as fast as possible.’ becomes ‘Tortoise.setSpeed(10);’. We include customized Javadocs with examples to support the student’s ability to discover the correct translation.

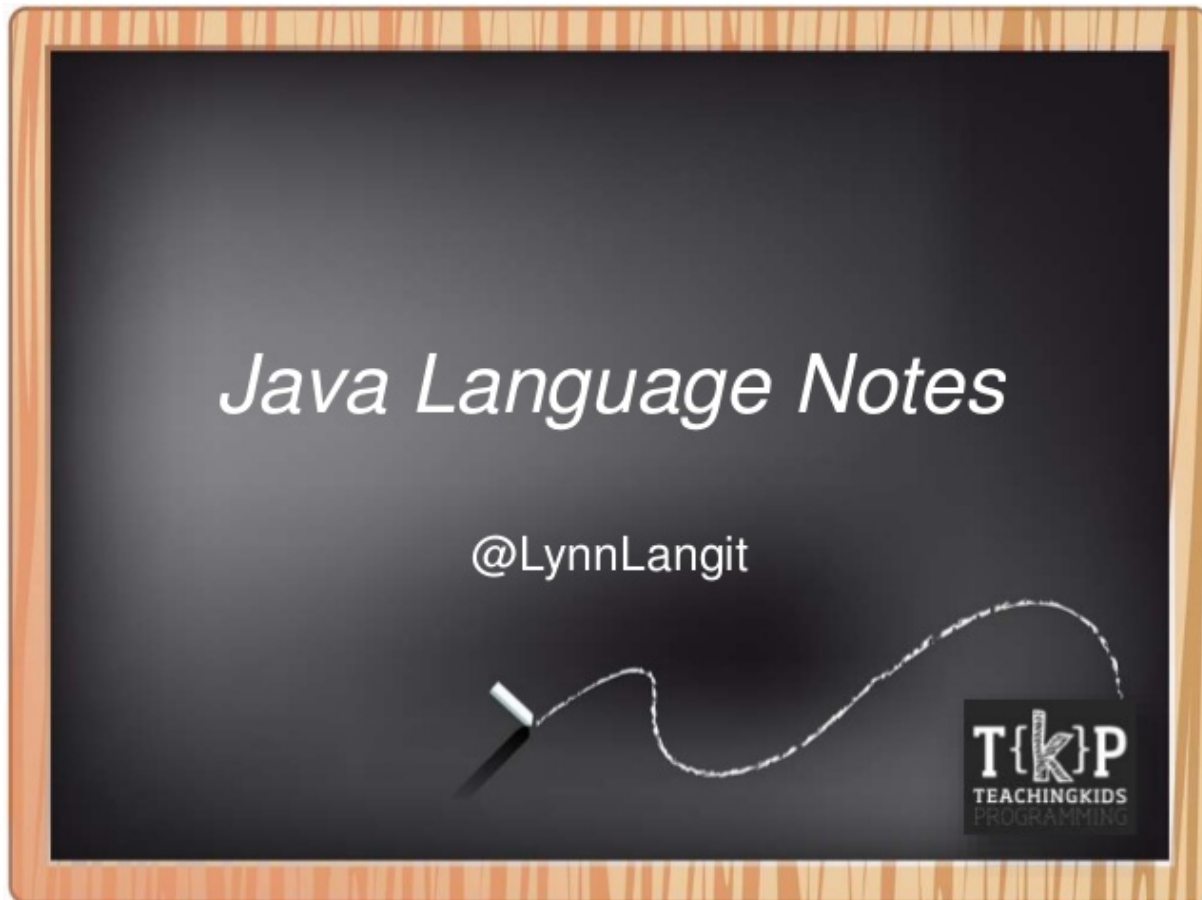
In some recipes we also deliberately introduce run-time errors and throw custom error messages, to further reinforce use of documentation while coding.

Recipes are supplied as a Java class file with method stubs which includes the English for each line of code that the pairs will write. Each Java class file also references pre-created Java objects, i.e. Tortoise, etc.. and is presented as one .java file per recipe, for example SimpleSquare.java.

:pushpin: See section on “Pair Programming” later on this page for more complete explanation.

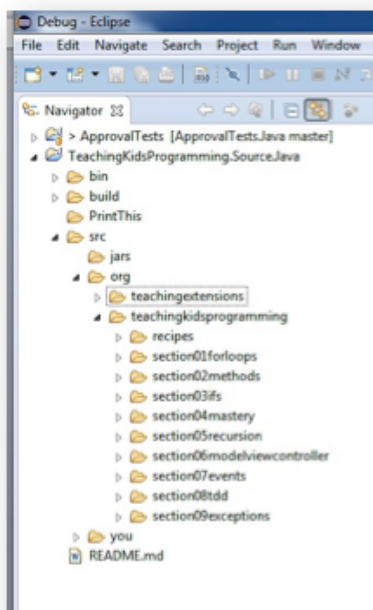
:pushpin: Teacher prepares to teach each course (in advance) by watching the TKP Recap YouTube video for the recipe that the pairs will complete. Teacher can pause the video and ask the students questions about the code in order to verify student understanding. Videos are found on the [TKP YouTube channel](#) and have the same name as the recipe, i.e. ‘Simple Square Recap’. Recap videos

are 10-15 minutes long. We also have created a simple PowerPoint deck which you can use to reinforce core Java language constructs (pictured below) - you can download it from [here](#).



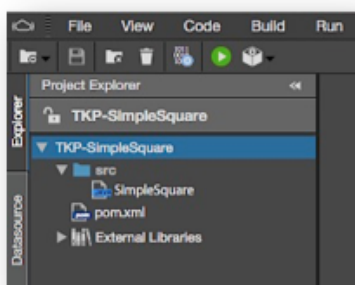
2) **Recap** – At this time students flip over their keyboards (or close their laptops) and observe as the teacher does the recipe the kids just coded him/herself. The teacher leads this section by using the Socratic method (i.e. asks the kids questions, such as 'What do I type now?' or 'What is keyboard shortcut for this action?', etc...) and codes the entire recipe again, while the kids answer the questions and watch.

:large_blue_diamond: Tip: Teacher should purposely make mistakes, such as adding extra parameter values, or leaving off closing braces, or omitting ending semi-colons and then ask the class 'How do I fix this error?' Also teacher should reinforce use of keyboard shortcuts, by asking students 'What is the keyboard shortcut to do action a, b, c (to delete a line, to complete a term, to perform a QuickFix, etc...). Below is a list of keyboard shortcuts for both Eclipse and Codenvy Java IDEs (editors).



Keyboard Shortcuts - Eclipse

Shortcut Action	Mac	PC
Auto-complete/Intellisense	Cmd+Space	Ctrl+Space
Quick Fix	Cmd+1	Ctrl+1
Delete the line	Cmd+d	Ctrl+d
Add a new blank line	Shift+Enter	Shift+Enter
Run the code	Cmd+F11	Ctrl+F11
Move a line of code	Alt+Arrow Key (Up or Down)	Alt+Arrow Key (Up or Down)
Cut	Cmd+x	Ctrl+x
Copy	Cmd+c	Ctrl+c
Paste	Cmd+p	Ctrl+p
Undo	Cmd+z	Ctrl+z
Save All	Cmd+Shift+s	Ctrl+Shift+s
Extract Method	Alt+Shift+m	Alt+Shift+m
Extract Local Variable	Alt+Shift+l	Alt+Shift+l
Rename	Alt+Shift+r	Alt+Shift+r
Find in project	Cmd+Shift+t	Ctrl+Shift+t



Keyboard Shortcuts - Codenvy

Shortcut Action	Mac	PC
Auto-complete Intellisense	Cmd+Space	Ctrl+Space
Delete the line	Cmd+d	Ctrl+d
Add a new blank line	Shift+Enter	Shift+Enter
Show JavaDoc	^j or Code>Doc	Ctrl+Q
Cut	Cmd+x	Ctrl+x
Copy	Cmd+c	Ctrl+c
Paste	Cmd+p	Ctrl+p
Undo	Cmd+z	Ctrl+z
Format Code	Code>Format	Code>Format
Open Declaration	F4	F4
Save all	Automatic	Automatic

3) **Variation / Kata Questions** – Teacher leads this section, students work in pairs. Teacher should ask students to name the properties of the completed recipes shape or output (for games), i.e. line

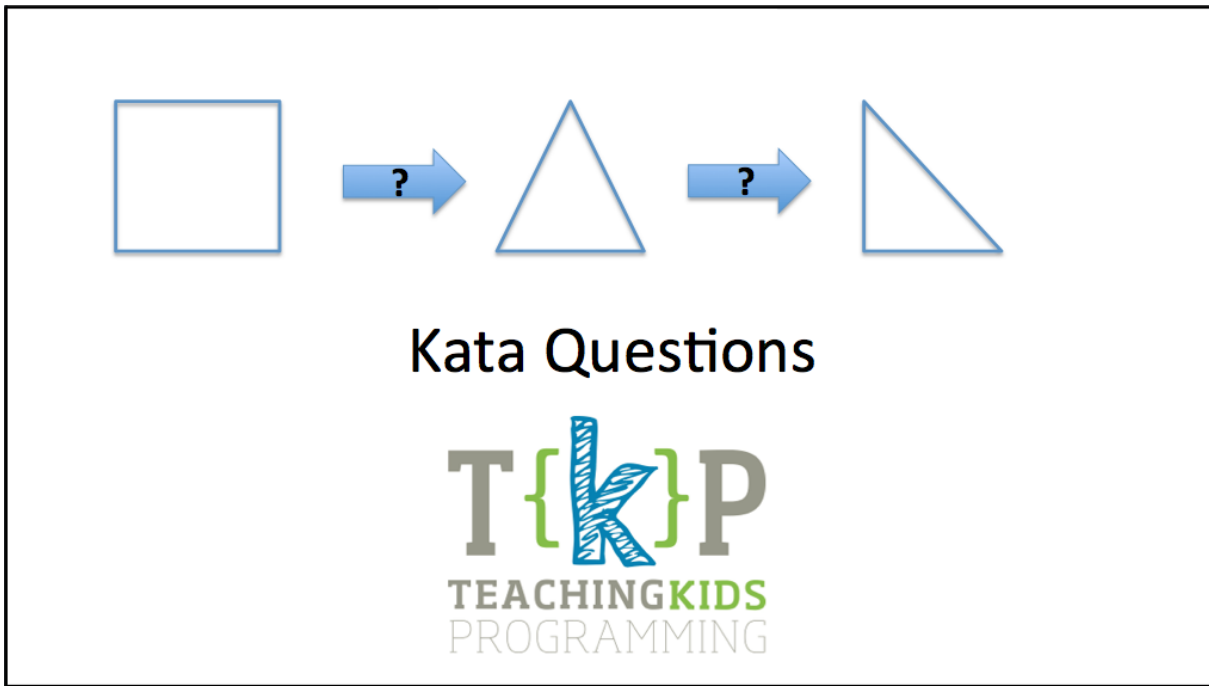
length, line width, etc...and should write on the white board as a grid (see example for SQUARE recipe below). Teachers should verbally guide pair of students in coding changes. Changes should start with simple changes, i.e. change variables from one constant value, such “blue” to “red”, then progress to complex changes, such as `Colors.Red` to `Colors.GetRandomColor()`. Written instructions on how to lead variations are found in each Course (Lesson) plan for the teachers. Teachers verbally lead students as described above.

:large_blue_diamond: Tip: Teacher can use one or more TKPJava ‘Kata Question(s)’ to ‘kick off’ variations. The idea (shown below) is to ask a question about the result of the recipe to get the students to visualize (draw), formulate (question), decompose (write and place English comments) and then to translate (code) to create the results.

TKP Kata Question Process



We’ve created a short video which elaborates on this process as well. It is found [here](#). Additionally, in each course we’ve added a ‘Kata Question’ at the beginning of the variation section.



Each course will include a sample variation grid, an example is shown below. The variation grid is meant to be a starting point for this section of the course. It is common, and desirable, that the pairs will experiment and explore beyond the basic variations that the teacher introduces.

Sample Variation Grid

Feature	Value	Refactor
Sides	5	no
Length	i	no
Color	ColorWheel	no
Rotation	1	no
Width	1-4	no
Background	SolidWhite (default)	expose default
Number of Lines	200	no

During the Variation portion of each course, the teacher reinforces code refactoring. TKP courses commonly include the following type of refactoring (used both when teacher is teaching the core recipes and also during variations):

Sample Types of Refactoring used in TKP courseware

Type	Example Usage	Keyboard shortcut
Rename	renames object	Alt+Shift+R
Extract local variable	expose default value	Alt+Shift+L
Extract local variable	make value variable and reusable	Alt+Shift+L
Extract method	pull out portion of existing method logic, usually for eventual reuse	Alt+Shift+M
Add comments	used to add comment markers	Ctrl + /
Inline variable	(advanced) used to improve code readability	Ctrl + 1

4) **Quiz** – Teacher should explain the format, i.e. translate one line, then run it to see if you got it right and then ask each pair to work together to compete the each line (or question) of the quiz correctly. Teacher is a **facilitator** in this section.

If the pair is stuck, then the teacher should guide the pairs to discovering the answer by having them re-read the line of English or to use the Eclipse editor to explore and to read the documentation (including the example usage of the objects shown in the documentation - such as `ColorWheel.addColor(Colors.Blues.Blue);`).

Teacher should encourage kids to run their program after they've translated **each** question. Teacher should remind pairs if they are stuck to use 'UNDO' to get back to a working state. Teacher should try to avoid telling students the correct answer, preferring instead to guide the pairs so that they can discover the correct answer themselves.

The quiz for a recipe is found in the CourseNumber folder of the TKP workspace, i.e. Course One w/Simple Square Recipe includes a Java stub file called Simple Square Quiz. Quizzes are named with the same name as the recipe, i.e. SimpleSquareQuiz.java and consist of a class with method stubs and English to be translated by the pair of students.

:pushpin: *Note: TKP Quizzes are NOT designed to be graded by teachers. They are designed to allow student pairs to demonstrate (and enjoy) concept mastery. Quizzes are designed so that ALL students can get ALL questions correct.*

5) **DeepDive** – Teacher should demonstrate only the first couple of deep dive questions as a group. In a deep dive, each method is a question. Teacher should encourage pairs to change **ONLY** the underscore characters when they are trying to get the method to pass its test. **IMPORTANT** Model running each test **FIRST** - it should fail and direct kids to read the failure output. Then change **ONLY** the underscore to make the test pass. Run that test to verify that it passed (result will be a 'green bar' in the test runner JUnit).

Teacher should use the same steps as for the Quiz to facilitate leading the kids through the DeepDive exercises. Students can work in pairs or in a Randori format (Randori is also called 'Mob Programming'). It is suggested to have the pairs rotate after getting each method to pass. If using a Randori format, the students should rotate after each successful test run. In either case, the teacher should encourage to share at least one thing s/he learned in the getting the method test to pass.

:large_blue_diamond: Tip: Here's more information about 'Mob Programming' - <http://www.agileconnection.com/article/mob-programming-whole-team-approach>

The deep dives are found in the Course Number folder of the TKP workspace for the current course, i.e. CourseOne, CourseTwo. They are numbered to match the TKP Course, i.e. Deep-Dive01.java, etc...they are designed to be run with the test runner in Eclipse and include English instructions at the beginning of the Java file on how to use keyboard shortcuts to run the file using the JUnit test runner.

6) **Worksheet** - TKP courses include printable (paper) worksheets. The goal of these handouts is to let teachers reinforce the key programming concepts (particularly terminology - such as classes, methods, variables, etc...) that have been introduced in each course by asking the students to identify or count the occurrences of particular types of those objects in the recipe solutions and record their answers on a written worksheet. You can download the worksheets from [this location](#).

7) **Xtras** – TKP courses have additional ‘mini-recipes’ included, such as the SPIRAL, which a teacher may use when teaching the SQUARE. These are 10-15 minute, fill-ins. They give the pairs who finish the guided work (quizzes, deep dives) before others something else to do. Teacher can have the pairs work on these ‘fill-ins’ on their own during guided teaching time or they can ask individual students to complete this supplemental work during free class time or at home.

Here is a list of all courses and materials. The list also matches computational concepts taught to TKPJava activities. And it lists the preferred method of teaching each activity (i.e. lead, facilitated, students in pairs, students in a mob, etc...). Items in yellow are in active development as of Jan 2016.

Course	Lesson/Recipe	Concepts Taught	teacher	students	Media
0-All	Concepts	All - Master Slide Deck	lead	n/a	Slideshare
0-All	Courseware	What's on the TKP website and courseware	watch	n/a	Penflip and video
0-All	Code	All - Answers	lead	n/a	GitHub
0-All	QuickTortoise	Demo recipe - for developers	Demo	watch	none
0-All	QuickShape	Demo recipe - for developers	Demo	watch	none
0-All	RecipeForTestingVirtualProc	Demo recipe - for developers	Demo	watch	none
1-For Loops	SimpleSquare	For Loops	lead	pair	Penflip and video
1-For Loops	SimpleSquareKataQuestion	Intentional Coding	lead	pair	video
1-For Loops	SimpleSquareVariation	Message Box, Extract Variable refactoring	lead	pair	video
1-For Loops	SimpleSquareQuiz	Using Objects / methods	facilitate	pair	Penflip
1-For Loops	Spiral	For Loops	coach	solo or pair	video
1-For Loops	DoubleLoop	Nested Loops	coach	solo or pair	Penflip
1-For Loops	DeepDive01	For Loops	facilitate	mob	Penflip
1-For Loops	Worksheet	Objects, Methods, For Loops	lecture	listen	none
2-Methods	Houses	Methods, Variables	lead	pair	Penflip and video
2-Methods	HousesKataQuestion	Intentional Coding	lead	pair	Penflip
2-Methods	Houses Variation	Methods	lead	pair	Penflip
2-Methods	Houses Quiz	Methods	facilitate	pair	Penflip
2-Methods	TriangleShell	Methods, For Loops	coach	solo or pair	Penflip
2-Methods	FourSquare	Methods, For Loops	coach	solo or pair	Penflip
2-Methods	DeepDive02	Variables	facilitate	mob	Penflip
2-Methods	Worksheet	Methods, Variables	lecture	listen	none
3-Ifs	HiLow	If/If Else	lead	pair	Penflip and video
3-Ifs	HiLowKataQuestion	Intentional Coding	lead	pair	Penflip
3-Ifs	HiLowVariation	If/If Else, Eclipse Debugger	lead	pair	Penflip and video
3-Ifs	HiLowQuiz	If, Message Box	facilitate	pair	Penflip
3-Ifs	ConcentricLoop	If, Nested Loops	coach	solo or pair	Penflip
3-Ifs	ChooseYourOwnAdventure	If, Methods, String comparisons	facilitate	mob	Penflip
3-Ifs	DeepDive03	If/If Else	facilitate	mob	Penflip
3-Ifs	Worksheet	Conditionals, program flow	lecture	listen	none
4-Mastery	DigiFlower	For Loops, Ifs, Methods, ColorWheel	lead	pair	Penflip and video
4-Mastery	DigiFlowerKataQuestion	Intentional Coding	lead	pair	Penflip
4-Mastery	DigiFlowerVariation	For Loops	lead	pair	Penflip
4-Mastery	PentagonCrazy	For Loops, Ifs, Methods, ColorWheel	coach	solo	Penflip and video
4-Mastery	PentagonCrazyQuiz	If, Methods, ColorWheel	facilitate	pair	Penflip
4-Mastery	BackgroundPhoto	For Loops, Chained Methods	coach	solo or pair	Penflip
4-Mastery	KnottedRing	For Loops, Methods	coach	solo or pair	Penflip
4-Mastery	DeepDive04	Types, type conversions	facilitate	mob	Penflip
4-Mastery	Worksheet	For Loops, Ifs, Methods, ColorWheel	lecture	listen	none
5-Recursion	TurtleTree	Recursive Methods, Use of HashMap	lead	pair	Penflip and video
5-Recursion	TurtleTreeKataQuestion	Intentional Coding	lead	pair	Penflip
5-Recursion	TurtleTree Variation	Recursion, Control Flow	lead	pair	Penflip
5-Recursion	TurtleTreeQuiz	Methods, Use of HashMap	facilitate	pair	Penflip
5-Recursion	SpiderWeb	Recursive Methods, For Loops	coach	solo or pair	Penflip
5-Recursion	RecursiveSquare	Recursive Methods, Chained Methods	facilitate	pair	Penflip
5-Recursion	DeepDive05	Collections, HashMap, Lists, Arrays	facilitate	mob	none
5-Recursion	Worksheet	Recursion, Control Flow	lecture	listen	none
6-MVC	AdLibs	String Concatenation, Message Box	lead	pair	Penflip and video
6-MVC	AdLibsKataQuestion	Intentional Coding	lead	pair	Penflip
6-MVC	ExceptionalAdLibsVariation	Conditionals, Regular Expressions	lead	pair	Penflip
6-MVC	AdLibsRtf	MVC with RTF parser	lead	pair	Penflip
6-MVC	AdLibs Quiz	Concatenation, Parser, Template	facilitate	pair	Penflip
6-MVC	OneFishTwoFish	Concatenation, Java Scanner Object	lead	pair	Penflip
6-MVC	DeepDive06	String Concatenation, Type conversion	facilitate	mob	Penflip
6-MVC	Worksheet	String Concatenation	lecture	listen	none
7-Objects	SuperTurtles	Objects, Instances, Constructor Method	lead	pair	Penflip and video
7-Objects	SuperTurtleKataQuestion	Intentional Coding, TKPJava Sounds Object	lead	pair	Penflip
7-Objects	SuperTurtlesVariation	Object Instances and Properties	facilitate	pair	Penflip
7-Objects	WhichFish	Objects, Instances, Case...switch statement	lead	pair	Penflip
7-Objects	CloneTurtles	Objects, Instances, Foreach Loops	lead	pair	Penflip
7-Objects	DeepDive07	Objects, Instance, Exceptions	facilitate	mob	Penflip
7-Objects	Worksheet	Objects, Instances	lecture	listen	none
8-Events	SimpleBubble	Objects, Instances, "this", Events	lead	pair	Penflip and video
8-Events	SimpleBubbleKataQuestion	Intentional Coding, Right mouse click events	lead	pair	Penflip
8-Events	SimpleBubble Variation	ArrayList	lead	pair	Penflip
8-Events	BubbleQuiz	Method chaining, events, Objects (Text...)	facilitate	pair	Penflip
8-Events	ConnectTheDots	Method chaining, events, Objects (Text...)	coach	solo or pair	Penflip
8-Events	TortoiseMaze	Button click event handlers	lead	pair	none
9-Kata/TDD	RectangleKata	Kata, Extract Method - refactoring	lead	pair	Penflip
9-Kata/TDD	WheelKata	Kata, Extract Method - refactoring	lead	pair	Penflip
9-Kata/TDD	FixzBuzKata	Kata, Extract Method - refactoring	lead	pair	Penflip

Here is a video of a talk w/code demos of TKPJava that Lynn Langit gave on TKP courseware design at Oredev in Malmo, Sweden in Nov 2014.

How TKP uses Professional Programming Practices

TKP instructional design is based on many Agile (and XP or Extreme Programming) best practices and principals. These include use of core Agile practices in the teaching (or delivery) of TKP courses. TKP courseware includes examples of using these practices in classroom situations and suggestions on how best to incorporate these practices into teaching children to program. They include the following:

- **Pair Programming** – both the students and the teachers work in pairs to learn / teach TKP material if possible. TKP courseware can be taught by a single teacher, however we recommend team teaching if possible. For the students, this means 2 students for 1 computer - one typing, one telling (what to do), rotate on time (usually 5 min) or task (i.e. when a line or section of code is completed successfully).
- **No Big Upfront Design** – teacher is advised on how to guide the student pairs to writing and executing their first program within 5 minutes of the start of each TKP class.
- **Test-driven development** - courseware is written so that students can be guided to translate one line of English into one line of code and then to run (or execute) the result, so that pairs can observe whether they have successfully completed the translation. This is a type of visual test-driven development. Also the TKP teaching method advocates deleting the original English comments AFTER the line has been properly translated.
- **Sustainable pace** – careful attention should be paid by the teacher to the pace of the class. Student pairs are rotated (within pair) either on task completion or on a regular time interval (such as 5 minutes). Pairs are also switched at the end of each lesson (if that day's class is a multi-lesson class), to further facilitate knowledge transfer. TKP courseware is designed to be taught via mastery-based learning - that is that every student masters every concept in each course, before the next course is introduced.
- **Rapid Feedback** - In addition to the immediate visual feedback that the children get after they run each line of code in the recipe, the TKP group advocates the use proctors (helpers) in the classroom to keep the pairs on pace. In addition to live proctors, the TKP courseware includes a Virtual Proctor, which provides visual feedback from all of the student pairs to the teacher.
- **Craftsmanship** – each recipe or lesson contains several sections, so that students can master concepts taught, APIs and tools before being introduced to new concepts, tools and APIs. This is also called 'mastery-based' learning as mentioned previously.

- **Randori** – a method of teaching a group with one computer and one projector. The group sits in a circle and rotates each time a task is successfully completed (coded). One person is at the keyboard typing (driver) and one person is telling the driver what to type (navigator). The group can assist the navigator if that person is unsure or ‘gets stuck.’
- **Code Katas** – a method of teaching by providing higher level abstractions to define the problem to be coded than a typical TKPJava recipe. For example:

```

1 // Here's the code for a kata
2 // Make a square
3
4 // Here's line -by-line instructions as in TKP recipes/lessons
5 // Show the tortoise
6 // Do this 4 times
7 //     Move the tortoise move 50 pixels
8 //     Turn the tortoise turn right 90 degrees

```

Code Katas more closely resemble real-world software requirements or user stories, than do line-by-line English instructions. The process of translating, or breaking down, higher level concepts into multiple lines of English comments FIRST and then writing the program code to execute each of those English instructions one line at time (and testing for correctness by running the program or a test) is key to the TKP Intentional method of teaching how to program.

- ****Code Koans**** – a method of teaching which uses both coding and retrospection (discussion AFTER successfully coding each method) to teach programming concepts, TKP uses koans (called ‘deep dives’) to explore programming concepts in further depth. We were inspired by the [Ruby Koans](#).

TKP Koans are based on the test-driven development (TDD) and unit tests. A unit test is a method which is written to compare the expected and the actual output values of another method. Unit Tests are often written with the ‘assert’ method. Another way to understand Koans is as a form of ‘learn by experimenting - literally student pairs are encouraged to ‘hack’ (or guess) to try to get the correct answer (or the unit test to pass [or produce a ‘green bar’]). A key component of working with TKP Koans is that the facilitator/teacher should ask each pair of students to reflect on what they have learned EACH time they get a unit test for a method to pass.

For example in the CLASSES TKPJava main recipe, the core concept of the ‘for’ loop is introduced. In the CLASSES/deep dive, the exercises include coverage of optional components of a ‘for’ loop, such as step value.

As they work through the koans, students will observe a variety of things about the language, environment, test harness, etc...teachers may want to capture student observations on a whiteboard, flipchart etc. during this part of each TKP course.

*

Why Pair Programming

Pair programming is a widely-used professional developer practice. Many studies have shown that this technique results in higher quality software (fewer bugs) AND happier developers.

Technique:

Pairing technique takes the form of one student at the keyboard and the other student sitting beside her. The student at the keyboard (*driver*) is taking directions from the student beside him/her (*navigator*). A simple timer (we often use our phone) is used to regulate the rotations, usually 5 minutes, but can vary, sometimes 'rotate on task completion' fits better. If there are two teachers, they should also rotate to model pair programming. Examples of alternative methods of rotating the student pairs, are rotating on task success (such as during the TKP Quiz or TKP DeepDive).

Methods:**Setup:**

Pairing allows twice as many students per computer. It reduces the amount of computers the teacher must set up, although it is very important to keep in mind the classroom will need the same amount of chairs compared to the number of students.

Class unity:

Unity is important because it allows the class to be on the same page. It is harder to have unity when students are working independently especially when it involves a large group of students. Pairing affectively reduces the amount of independently working students by $\frac{1}{2}$.

Skill equalization:

Students come in with different skill levels, making it difficult for a teacher to teach them simultaneously. The teamwork pairing offers helps to level the playing field for all the students. The best part is that it does not penalize the more advanced students because as they explain the material to their classmates, they gain a deeper understanding.

Focus:

Working with a partner helps both students to keep their energies focused on coding and reduces distractions (i.e. web browsing, etc...).

Social:

Pairing offers a more friendly and sociable atmosphere, which we hope to dispute the misconception that programming is associated with being anti-social. This is especially important for girls, who usually work better in groups where they are able to communicate and discuss with one another.

Reduces frustrations:

In programming, encountering bugs cause a huge amount of frustration for everyone. When students are individually working, it is easy to overlook bugs or mistakes. However, with two pairs of eyes attentively working together, the amount of bugs or errors present are greatly reduced. Often times a concept may be confusing to one of the students in the pair, but not the other. By working together their frustrations are reduced.

Body Chemistry:

Rotations are implemented by the teacher because after 15 minutes of sitting down, one's body chemistry shifts and slows down. By requiring the kids to **stand up** every 15 minutes, we help prevent this shift in their body from taking place. This is the reason behind why the teacher should refrain from just letting the students pass the keyboard over when it is time for the pairs to rotate.

Greater peripheral vision:

According to cognitive science studies, negative emotions hinder one's ability to engage in broader visual perception. (<http://www.unc.edu/peplab/broadening.html>)

For example, a study tracked the duration of time people spent looking at a picture according to their emotional state. The more unhappy one is feeling, the less likely the individual will be able to see the entire screen. It is difficult to learn when one is not able to be able to see options available to them.

Creativity:

Studies in cognitive science have also presented the fact that one's creative abilities may be limited when working in solitude. (<http://ascc.artsci.wustl.edu/~ksawyer/grouppgenius/>) However, creativity is able to flourish when one is given the opportunity to bounce ideas off others.

During class

Kids all working in pairs unless otherwise noted. At 5 minute intervals (use a timer), pairs get up switch roles. One types, while the other tells the typist what to do. An example is shown from a TKPJava class below.

:pushpin: Below is an example of two kids pairing while learning TKPJava.



Silicon Valley Code Camp, Kids Track, Oct 6, 2013 (Arun Gupta)

*

Why Group (Mob) Programming

Another method of teaching we use is group (or mob) programming. This is a process with one computer, which is attached to a projector. One student sits at the keyboard ('the driver') and types what one other students who is standing ('the navigator') tells that student to type. The rest of the students sit in a circle of chairs around the driver and navigator.

For each coding problem, first the navigator tries to solve it. The driver types what he or she is told to type and then runs the program to verify that the result is correct. Next the teacher (who is acting as a facilitator during this process) asks BOTH the driver and navigator to share at least one aspect of coding, tools, language, etc... that both learned by solving the problem. The facilitator can optionally record the student's information on a whiteboard or flip chart.

After a problem is solved, then the students all **stand up** and rotate in the circle, so that every student will have at least one turn as driver or navigator during the exercise.

For more about the benefits of mob programming, see this reference – http://en.wikipedia.org/wiki/Mob_programming

*

TKP Classroom Management Tips and the TKP Virtual Proctor Utility

We included this section for teachers of our courseware who are new to leading a classroom full of kids. Experienced K-12 teachers may wish to skip this section as they probably are familiar with the majority of these techniques.

Small steps

- Translate the next line that you can see the result of execution – you will guide them through the recipes in the order that can SEE the results, NOT in 1,2,3 order. All of our recipes have suggested line orderings on the English lines for teaching, i.e. teach line 2 first, teach line 5 second, etc...
- If a single line isn't viewable, use 'Fake it till you make it' to allow interim translations i.e. Place the Circle where the mouse is (you don't have a circle, which needs a radius, or a mouse) `ShapeMaker.CenterShapeAt(ShapeMaker.CreateCircle(20), 20,20)` "ok, this puts a circle on the screen, now lets work on it being the correct circle, and using the mouse locations..."

Instructor typing

- Always type *just* enough to show the documentation that would be the most helpful to the kids.
- If they are looking for something that the tortoise should do, type so that the list of methods comes up on the side.
- Give the kids time to explore the documentation.
- Always wait until most of the pairs have got working code (look at virtual proctor results) before the instructor types out the code and runs it.
- It's ok to be quiet for a while and let the kids work. No, really.
- It's great to occasionally make a mistake when the instructor types, either by typing exactly what the example/documentation says and then correcting it because what we need is slightly different (`SetPenColor("red")` / `For i = 1 To 10`), or to get something that doesn't compile (e.g. during the recap, try typing `Tortoise.Turn(left)` and then when it doesn't compile, point out the line number and the helpful error message, and then to the documentation on the side again).

Guide — don't tell

- Ask the kids to read the English lines out loud – use this technique frequently.
- If kids are stuck, have them re-read the relevant line in English.
- Exploration, have the kids use the arrows to figure out what to do. If they don't know where to start `Ctrl+Space` will list all possibilities.
- If someone asks "is this right?", tell them "run it".
- If they want to know which method, point out the documentation.
- Resist the instinct to tell them the right answer, but help them to keep moving.
- Watch to see both members of the pair are contributing, if not, prompt the non-participating member of the pair to 'jump in there'

Create and use save points

- At least at the end of each recipe, everyone must save their work

- To revert to the beginning of a lesson quickly, go to the Navigator in Eclipse, right click on the Java file and then click on 'Revert using local history' and then click on the first entry, this will re-set the lesson back to the start point

Recaps

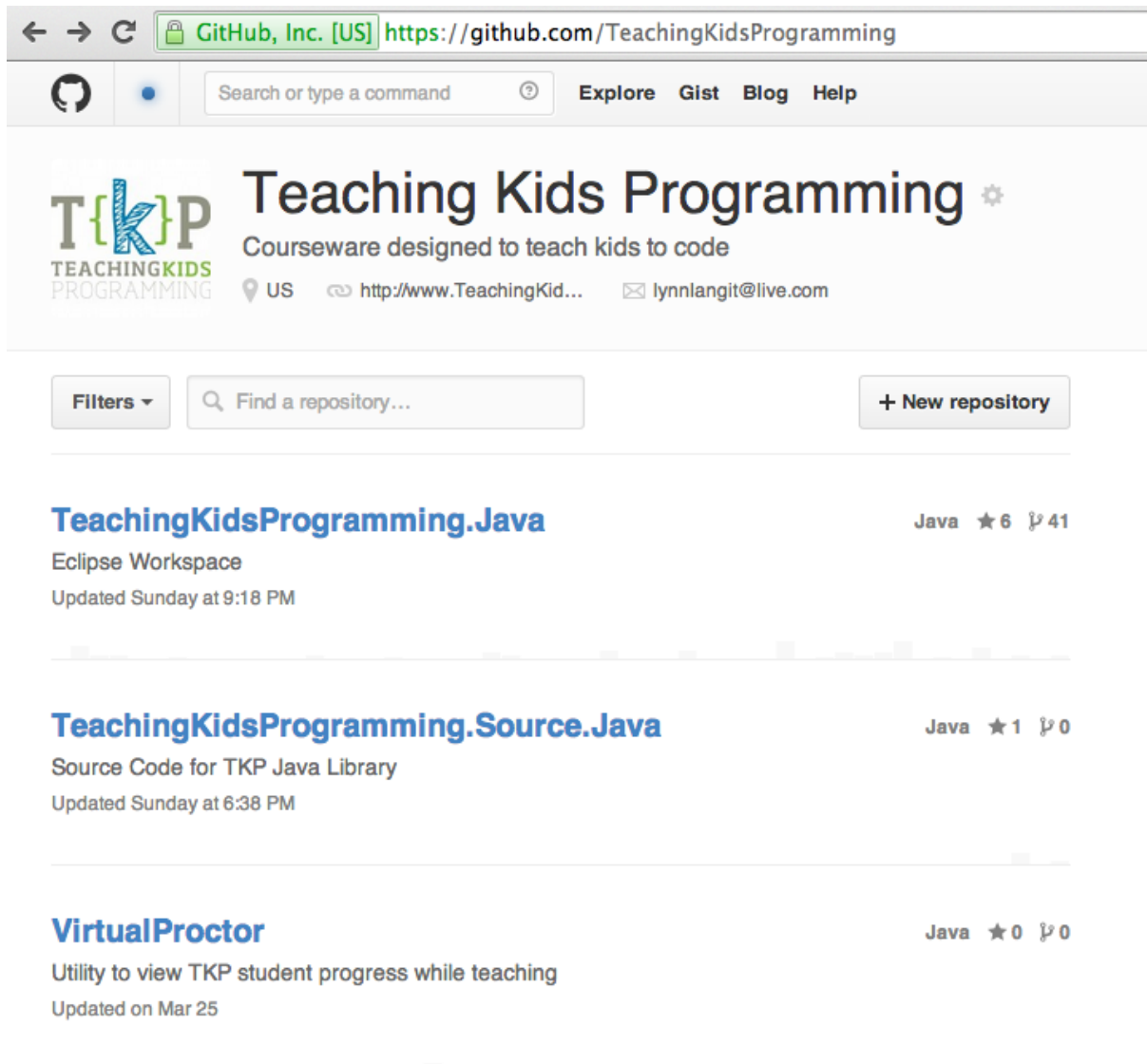
- Point out nuances of the editor, the documentation, the logic, the shortcuts as you go.
- Use this time to Explain the Concepts they have just Experienced.
- Introduce Programming Lingo, .i.e for SQUARE say 'nouns are Objects and are shown with Cubes; verbs are Operations and are shown with Wheels, etc...
- This is a good place to do a scoping diagram (list Objects, Methods and Variables you have used)
- Here is a good place to make mistakes on purpose, such as "Tortoise.SetPenColor (blue)" and have the kids tell you how to correct those mistakes

```

1  ##### Coding (Debugging) Tips
2  - UNDO: Advise the use of undo first
3  - FORMAT: we have set Eclipse to automatically format on save.
4  - METHODS (Operations): check parenthesis pairs and arguments - have the kids
   click on the method (word) in the editor and then read the documentation -
   particularly the example.
5  - BLOCKS: check for { } placement and/or missing { }
6  - STRINGS: check for double quotes
7  - ENDLESS LOOPS: undo back to a runnable state, this is usually a result of
   putting a string in the arguments for a loop
8  - DEBUGGER: Use the Eclipse Debugger - here's a short YouTube screencast on how to
   do this - https://www.youtube.com/watch?v=M8F6ziuccmw
9
10 ##### Pairing & Energy
11 - Adjust "double-"slow pairs with fast kids; make sure to rotate a pair that falls
   behind continually. Do not pair siblings. If adults want to take the class,
   pair the adults together.
12 - Backseat Drivers: if a kid who should be navigating is typing, then interrupt,
   and remind the kid that they can NOT touch the keyboard when they are
   navigating
13 - Rotate teachers same as kids
14 - Rotate every 5 minutes or at task completion
15 - Make them stand up when rotating, this simple movement helps with energy and
   attention.
16 - Kids should be typing within 3 MINUTES of class starting. It is very important
   to start the class off with action quickly.
17 - Ask the kids LOTS of questions, all of the time. 'Dont tell them, ask them -'
   its critical to keeping them engaged.
18 - Encourage barely controlled chaos - noise is good, students should
   be talking/laughing;
19 - During Variations kids should be joyful.
20 - If teaching after lunch, spend extra time pumping up the energy before sitting
   down.
21 - Switching seats should be a high energy activity. Do NOT raise hands BEFORE
   doing this, as it lowers the energy in the room.

```

Guide to TKP Java courseware and source code on Github



The screenshot shows the GitHub interface for the 'Teaching Kids Programming' organization. The header includes the GitHub logo, a search bar, and navigation links for 'Explore', 'Gist', 'Blog', and 'Help'. The main content area displays three repositories:

- TeachingKidsProgramming.Java**: Java, 6 stars, 41 forks. Description: Eclipse Workspace. Updated Sunday at 9:18 PM.
- TeachingKidsProgramming.Source.Java**: Java, 1 star, 0 forks. Description: Source Code for TKP Java Library. Updated Sunday at 6:38 PM.
- VirtualProctor**: Java, 0 stars, 0 forks. Description: Utility to view TKP student progress while teaching. Updated on Mar 25.

The courseware is open source. The teaching version (to be installed on the student computers, consists of a Java .jar file and some utility files. The teaching version of TKP courseware is found in the Github :octocat: repository named [TeachingKidsProgramming/TeachingKidsProgramming.Java](https://github.com/TeachingKidsProgramming/TeachingKidsProgramming.Java).

About the TKP Virtual Proctor Utility

In addition to the courseware, TKP has two related repositories on Github. The first one is named [TeachingKidsProgramming/VirtualProctor](https://github.com/TeachingKidsProgramming/VirtualProctor). This is a simple application that we use to help us to monitor the progress of the students in the classroom. To use this utility, just open a browser to <http://virtualproctor.tkpjava.org>

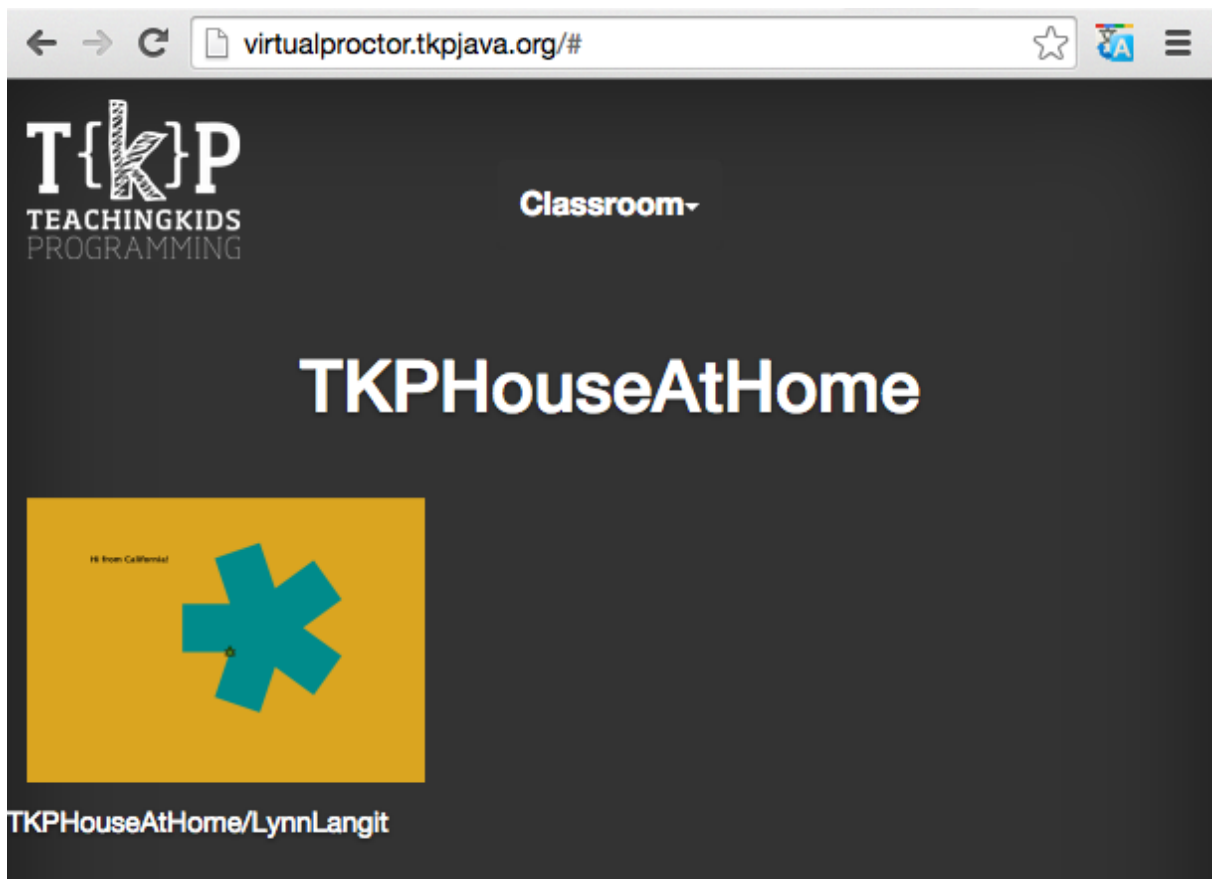
By default the TKP VirtualProctor takes a screen shot of the program (graphics) windows AFTER the student closes the window and sends that screenshot to a cloud-based service. The associated

website displays the last screenshot with the default name of the computer.

To configure the VirtualProctor with a customized student name from a student's machine, add the following line of code to any TKP recipe:

```
1 VirtualProctor.setName("studentsName");  
2 VirtualProctor.setClass("classroomName");
```

After you add this line of code, then run the code and close the program (graphics) window. The TKP Virtual Proctor contains an automatic refresh interval. You can drag and drop the screens around on the browser page so that the screens match the physical classroom layout. A sample output is shown below.



The source code for the APIs that we wrote to use in the TKP courseware is found in the Github repository at [TeachingKidsProgramming/TeachingKidsProgramming.Source.Java](https://github.com/TeachingKidsProgramming/TeachingKidsProgramming.Source.Java). Here you can see which APIs we wrote (or extended) in creating the courseware. Also of interest to teachers in this repository may be [the completed recipe \(or answer\) section](#). We deliberately obscured the location of the answers, as we do not want students to copy and paste the answers, rather we'd prefer that they explore the API, and with teacher guidance, code the results themselves.

*

Guide to Writing TKP Writing Recipes

We put a great deal of effort into creating recipes. We test and iterate constantly, so that kids will have a great experience learning. Some of the core concepts we try to follow when creating recipes are as follows:

- 1) Recipes should be FUN and ENGAGING
- 2) Recipes should be able to be completed in 15-20 minutes
- 3) Recipes should be around 20-40 lines of English (and no longer)
- 4) Recipes should teach 1-3 core programming concepts only
- 5) We create both the recipe and the variation (often also the quiz) when we create a new recipe
- 6) The process goes as follows:
 - get an idea, for example draw an object, write something to the screen, etc...
 - code it - see what objects, properties, methods or events exist in the language already
 - create some new (or wrapped objects) in the Java TKP library (on Github), such as 'Tortoise.SetPenColor' to encapsulate 'ProgramWindow.PenColor' to make lesson objects more discoverable for kids
 - write documentation, including code examples for your new objects into the new TKPJava objects, methods, properties, etc..
 - create the 'code stub' for the recipe, that is stub out the main method and any other methods that you just want the kids to fill in.
 - add the English comments to your stubbed out recipe
 - determine the 'run order' and add the line numbers to the end of each line of English
 - test the recipe with actual kids!
 - make changes and updates based on your tester (kids) feedback
 - edit the TKP Java source code on Github (you'll get your own branch to update)
 - add your new recipe to your TKP branch and create a pull request so that we can see your recipe
 - create teacher training materials (cheat sheets [answer keys] and videos)
 - add information to the TKP Penflip course number about the how to teach the recipe

Recipe Writing Guidelines (for the English Comments in Java Recipes)

thanks to contributor Katrina Owen for this guide

'recipe formatting notes

'add empty lines after each instruction

'add two empty lines before starting a recipe

'add spaces after the apostrophe to indent comments

'indent method four spaces

'indent loops four spaces

'always end loops with 'Repeat

'start a method recipe like this:

'_____ Recipe for SomeRecipeName

'end a method recipe like this:

'_____ End of SomeRecipeName recipe

'use English, not code, e.g. instead of 'Set the PenColor to "ReallyBrightRed"

'use 'Change the color of the line the tortoise draws to really bright red

'corollary:
 'do not capitalize object names in comments
 'do not put scare quotes around color names
 'lower case color names (and do not string them together)
 'start variable names with 'the current x is y'
 'always format program at the end to get rid of whitespace on blank lines
 *

TKP courseware influences / influencers

We have been asked to create a list of influencers/influences on TKP courseware - here is a link to a Gist with that information – <https://gist.github.com/lynnlangit/15e12d902ba66654468b>

TKP courseware for other computer languages - SmallBasic, C#, T-SQL and more

Our main focus is to write courseware for middle school aged kids in Java. However, we have experimented with other computer languages over the years. We use our 'TKP Intentional Method' of teaching in every domain. To that end, we'll list links to our other courseware repositories below:

- 1) SmallBasic on [Codeplex](#) and on [YouTube](#)
 - 2) C# on [Pluralsight](#)
 - 3) Kodu on [Slideshare](#) and on Lynda.com (<http://www.lynda.com/search?q=kodu>)
 - 4) T-SQL on [Codeplex](#)
- *

About TKPJava and the APJava CS Exam

Via teacher feedback we at TKP hear that our TKPJava courseware is being used as part of the preparation for students prior to them taking the the APJava CS exam.

While we do not design our courseware to map to the exam objectives specifically, we will provide a chart (below) to list the exam concentration areas and our course objectives.

[Here](#) is a more complete list of the AP exam objectives.

Exam objectives – Goals of AP Computer Science A

Students should be able to

- Design, implement, and analyze solutions to problems;
- Use and implement commonly used algorithms;
- Develop and select appropriate algorithms and data structures to solve new problems;
- Write solutions fluently in an object-oriented paradigm;
- Write, run, test, and debug solutions in the Java programming

language, utilizing standard Java library classes and interfaces from the AP Java subset;

- Read and understand programs consisting of several classes and interacting objects;
- Read and understand a description of the design and development process leading to such a program; and
- Understand the ethical and social implications of computer use.

*

Topic Outline for AP Computer Science A**

Object-Oriented Program Design - some coverage in TKPJava

A. Program and class design - Course 6-8

Program Implementation - some coverage in TKPJava

A. Implementation techniques - all Courses

B. Programming constructs - all Courses

C. Java library classes and interfaces included in the AP Java Subset - partial

Program Analysis - some coverage in TKPJava

A. Testing - Course DeepDives and Course 8

B. Debugging - optional, can be included as teacher-led

C. Runtime exceptions - Course 6

D. Program correctness - all Courses

E. Algorithm analysis

F. Numerical representations of integers - DeepDives and Courses 6-8

Standard Data Structures - covered in TKPJava

A. Primitive data types (int, boolean, double) - Courses 1-8

B. Strings - Courses 1-8

C. Classes - Course 7-8

D. Lists - Courses 5 and up

E. Arrays (1-dimensional and 2-dimensional) - Courses 7 and up

Standard Operations and Algorithms - some coverage in TKPJava

A. Operations on data structures - Course 7 TKPJava

B. Searching

C. Sorting

Computing in Context - not covered in TKPJava

A. System reliability

B. Privacy

C. Legal issues and intellectual property

D. Social and ethical ramifications of computer use

*

2 Teaching TKPJava Course 01 - Intro to Objects

Objects, Methods and For Loops
*

Preparing to Teach this Course

Every Course

:hourglass: **Install** the [TKPJava courseware](#)

:green_book: **Read** this lesson plan page

:computer: **Code** all recipes yourself

:bulb: **Review** the [TKPJava Language pptx](#)

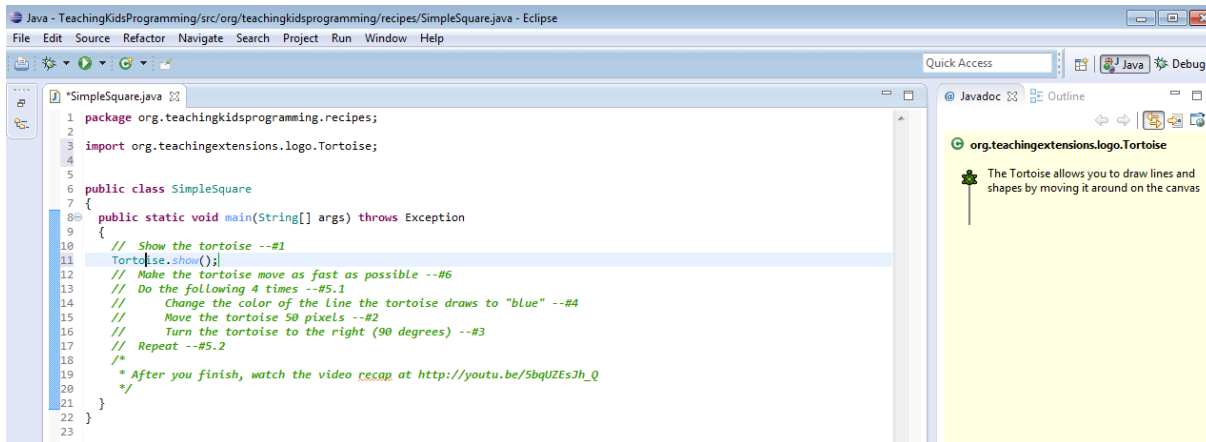
:fax: **Print** the [keyboard shortcut sheet] (<http://www.slideshare.net/lynnlangit/tkpjava-eclipse-and-codenvy-ide-key>)

:fax: **Download and Print** the [recipe worksheet] (<https://www.dropbox.com/s/9qwby48p8lmx4nj/TKP-Worksheet>)

:bar_chart: Keep Learning!

Initial TKPJava Courseware Installation

- Install the TKPJava courseware (on GitHub :octocat:) – TeachingKidsProgramming. by following the one page detailed installation instructions for PC or Mac - [here](#).
- After downloading per the instructions above then double-click the launcher file (**TKP_launcher.jar** - for Windows) to open the customized version of Eclipse to the TKPJava workspace.
 - If Eclipse doesn't open at all, you need to install Java or Eclipse or both. Re-read the instructions (on GitHub) to understand how to manually install Java and/or Eclipse plus the TKPJava courseware on either a PC or a Mac.
 - Shown below is an example of how the TKP customized workspace **should** look if the TKP_launcher ran correctly.



Part 1 - Recipe: Simple Square

As this lesson is the beginning of the course, it's quite important for the instructor to set the proper tone for the student work. TKPJava is instructor-facilitated via interrogative guidance.

- To start teaching, pair the students (see 'pair programming' in the previous Penflip course), then have the students read the first line of English (comments) in the SimpleSquare.java code file ("Show the Tortoise" – #1) out loud. Line numbers are at the end of each line.
- Use the TKP Intentional method to guide students so that THEY can do the translation. Refrain from telling them the answer (Java code) in advance.
- Encourage students to use 'ctrl+space' to get a list of possible items rather than typing out the Java. Note that **Java is case-sensitive**.
- After the students complete translating each line of English into Java, it is very important to **MAKE SURE THEY RUN THE CODE BEFORE PROCEEDING**.
- A detailed example teaching script for the first line is shown below.

LINE NOTES

To get started, we've provided a complete example (teaching script), to give teachers a blueprint for facilitating TKPJava in the Intentional style.

Say: "Read Line #1 out loud". ("Show the Tortoise")

```
1 // Show the Tortoise --#1
```

Ask: "What is the noun in that sentence?" (Tortoise)

Say: "Move your cursor to the end of that line and click, to put the cursor there."

Say: "Press and hold the 'shift' key plus the 'enter' key to add a new blank line below the line you are on now for your Java code.."

Say: "In that new line type Big 'T' then small 'or', then press and hold ctrl+space."

```
1 // Show the Tortoise --#1
2 Tor
```

Ask: "Do you see the word 'Tortoise' in the drop-down list?"

Say: "Click on that word in the list and press enter to add the Tortoise"

```
1 // Show the Tortoise --#1
2 Tortoise
```

Say: "In English we separate words with a space, in Java we use a period. Type a dot after Tortoise."

```
1 // Show the Tortoise --#1
2 Tortoise.
```

Ask: "What is the verb in that sentence?" (show)

Ask: "How do you get your list?" (ctrl+space)

Ask: "Do you see the verb? Notice it starts with small 's'" (show)

```
1 // Show the Tortoise --#1
2 Tortoise.s
```

Say: "Click on the verb in the list and then press enter to add the verb." (show)

```
1 // Show the Tortoise --#1
2 Tortoise.show()
```

Say: "How do we end a sentence in English?" (with a period)

Say: "In Java, we use a semi-colon. Type that to end your sentence."

```
1 // Show the Tortoise --#1
2 Tortoise.show();
```

Say: "It is important to make sure you've translated correctly after each line.

To test it out, you have to run your program. To run your program click the (green) play button."

WAIT

Ask: "What did you see?" (A Tortoise!)

Say: "Click the 'X' in the upper right corner to close the window."

Say: "Now we have to clean up our work. To clean up we will delete the English line that we just translated."

Say: "The easiest way to delete that line is to click anywhere in that line, then ctrl+D." _

Have the students read Line #2 out loud.

Follow a similar process to the detail listed above for Line #1, i.e. ask questions, etc...

```
1 Tortoise.move(50);
```

Remind them to run to verify and then to clean up the English that they have correctly translated.

```
1 Tortoise.turn(90);
```

Explain setters and getters - you may wish to show the Getters/Setters slide from the TKPJava-Language.pptx deck to anchor the concept.

Say: "Setters change something about an object; getters tell us something about an object."

Choose a student to be an object.

Provide an example, i.e. "Set a student's position to standing / Get a student's hair color."

`Tortoise.setPenColor(PenColors.Blues.blue);`

NOTE: The reason for the use of 'PenColors' rather than 'Colors' or 'Color' is that there is a 'Color' library in standard Java. The TKP 'PenColors' library includes custom documentation and examples of using the object. The documentation includes 'color swatches'. There are also several other Java libraries that include a 'Color' or 'Colors' object. Having the students select 'PenColors' guides them to using objects in the TKPJava PenColors library.

For Loop

Say: "Read the text at line -#5.1" (// Do the following 4 times)

Ask: "Do you see a line numbered #5.2? What is that for?" (repeat - ends the loop)

Ask: "Which line numbers (of code) need to be repeated?" (line x, y, z...)

Ask: "What is the easiest way to repeat?" (copy and paste four times)

Say: "What is the keyboard shortcut to copy? to paste?" (ctrl+c, ctrl+v)

Have the kids copy and paste the lines 4 times and then run it, then undo back to line

Ask: "But what if you had to do it 400 times?"

Say: "Copy/paste over and over, violates the most important rule for programmers – (use your tools). We don't want to break that rule. So..."

Introduce 'for' loop syntax, you may want to use the TKPJava.pptx slide

To translate, type "for" and then ctrl+space, select the first "for loop template"

- "array.length" needs to be replaced by the number of iterations (4)

- DELETE the second line of the array template

- Move the end bracket down **after** the three lines of code. Use alt+down, to move the end bracket to bottom of loop code.

```
1 for (int i = 0; i < array.length; i++) { //loop body here }
```

Tip: Make sure they run it to verify. Check – easy to mess this one up!

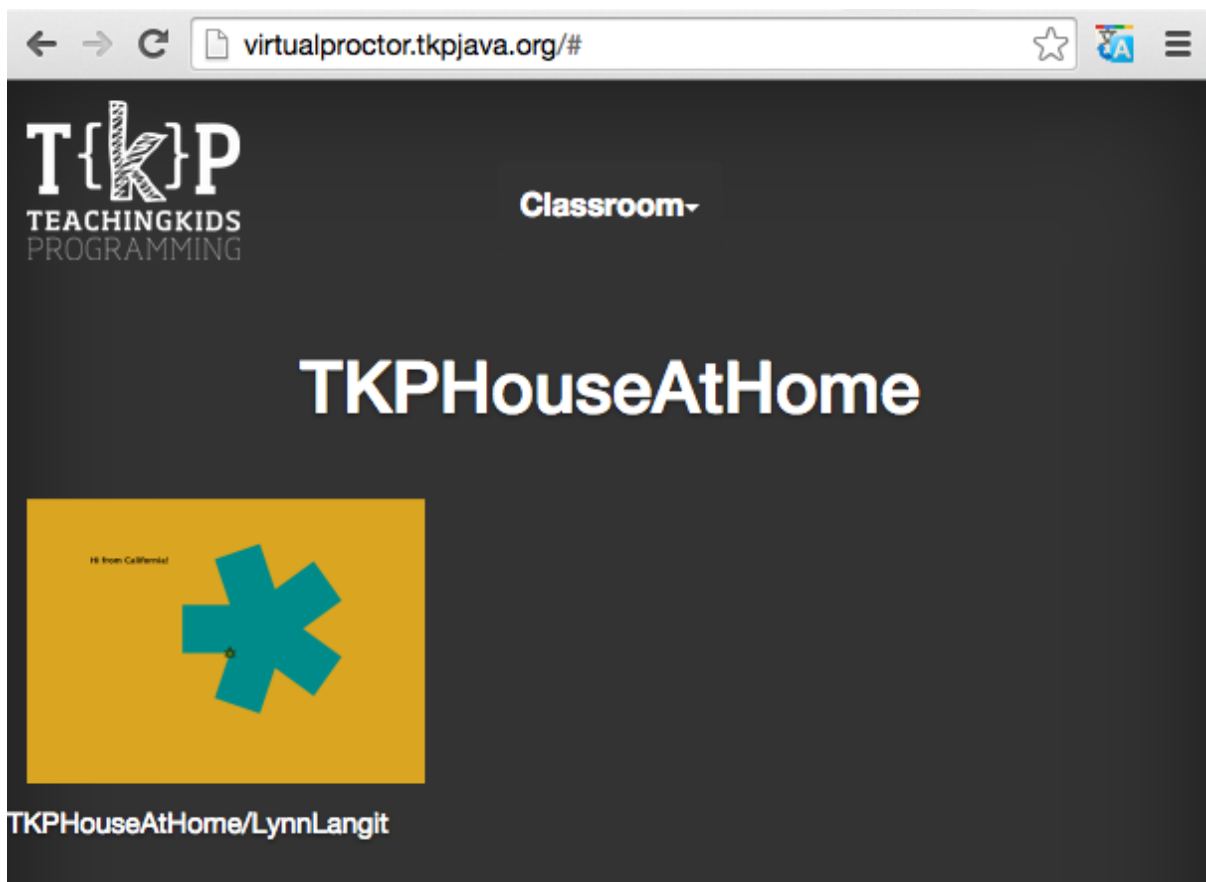
After eliciting the command, ask: how do we know what is "as fast as possible"?

Have the students click on the 'setSpeed()' method and direct them to look at the example in the Javadoc window to see how to assign the appropriate value for speed (10).

```
1 Tortoise.setSpeed(10);
```

CONGRATULATIONS! You just made your first program!

:large_blue_diamond: *Tip: You can use <http://virtualproctor.tkpjava.org> from any computer to show student windows on screen when they close their window. We like to run the TKP Virtual Proctor from the instructor's computer and project the screen so that the kid's can 'see' what they are all coding (example shown below).*



Here are some optional lines to personalize the view in the Virtual Proctor

```

1
2 // NOTE: see your work at http://virtualproctor.tkpjava.org/
3 // Set your classroom name for the Virtual Proctor --#6.1 (optional)
4 VirtualProctor.setClassName("TKPworld");
5 // Set your student name for the Virtual Proctor --#6.2 (optional)
6 VirtualProctor.setName("TKPstudent");

```

Part 2 – Variation: Simple Square

To kick off this section, you may choose to use the TKP Kata Question:

“How would you make a triangle here?”

TIP: Use the ‘Save as...’ file name feature to save your work for each type of triangle, i.e. scalene, equilateral, etc... you make.

See the TKP Instructional Design (Teaching Variation) section for more information.
Here are some sample kata questions - examples [here](#).

Here's a graphic that represents the general idea and [here is a short screencast](#) on how to teach code mini-katas:

TKP Kata Question Process



This is lead as a 'follow the leader' style exercise. Open Word or something to scribble in. Draw a feature grid on your whiteboard as shown below. Ask your students – what features are there? Example shown below - (with all later steps completed):

Feature	Value	Prepare	Change 1	Change 2
Angle	90	Expose Math	360 * 3	Negative
Color	Blue	Introduce Colors Object	Purple	Randomize
Width	2	Explicit Default	8	i * 1.5
Sides	4	Extract Variable	7	Ask User (Message Box)
Length	50	Nothing	23	i * 4

[Here's](#) a screencast (video) on how to teach the SimpleSquareVariation. We've made these video for teacher preparation:



Simple Square Variation



Variation Preparation

Intro to preparation

Any of these values can be changed. But first, we have to get the code ready to be changed, or it'll end up a mess!

[Add another column – prepare. Must do these before moving on to other things.]

- Prepare angle
- Hard to change right now
- Go to 90 in code. $90 = \text{quarter of circle}$. (Spin around.)
- How do we know that 90 is $\frac{1}{4}$ of 360? Math. (Or memorization.)
- Computers better at this than us. What if we want it to turn $\frac{1}{7}$ of a circle? What is $360/7$? Uhh...
- Rewrite as $360/4$
- Run, see that it's the same. [THIS IS IMPORTANT.]
- Put X in “prepare” for angle.

Prepare sides

Simple refactoring. Go to where it says 4, highlight, right click, refactor, extract local variable.

Name it “sides”

Run – show that it does exactly the same thing.

[Don't explain everything here – just show that it runs.]

X it off in the graph.

Prepare line width

Which line determines this? None. It's a default. So we're going to make the default explicit.

FIRST LINE IN for loop, add: `Tortoise.setPenWidth(2);`

Go back to grid, X it off

Other Variation Notes

[Overall note for variations: try 3-5 numbers, some ridiculously small, some ridiculously big. Don't use "normal" numbers (20, 100) – use 13, 117, etc.) Don't go over 1000.]

New column: "Simple changes"

Add one more column: "Cool changes"

Can change anything, but needed to prepare before we mess around.

Variation: sides

Change sides to 7.

Change sides to 13.

Change sides to 100.

Anyone notice a problem?

Lines don't meet at a certain point. Why? What is 360/13? Fraction!

Java doesn't automatically deal with fractions... so change 360 to 360.0 so we're telling it we're dealing with fractions. NOTE: You are introducing the concept of a Double vs. an Integer.

Variation: colors

No refactoring – show green, purple, let them pick a couple of favorites.

Show swatches of color as they arrow down.

Have them find a way to get a random color

`Tortoise.setPenColor(Colors.getRandomColor());`

Variation: line width

Start with a few small – 5, 8, 13, 75, 347, bring it back to 7

Variation: Move length

99, 9, 23. Check it off. (Just quick changes here.)

Cool changes

Every time we run this it does the same thing. Now, let's make it so user tells us how many sides to include.

Replace sides = 7 with `MessageBox`

```
1 int sides = MessageBox.askForNumericalInput("How many sides?");
```

Run it – show a few options

Show that sides is in black. So is i. These are variables.

In "setPenWidth", change 1 to "i".

Ask: what do they think will happen?

Talk them through what's happening. How wide is line 1? What about the fifth line? The (next one after the largest?) – there isn't one.

Do a few more – `i * 10`, `i * 3.5`

```
1 Tortoise.move(i*5);
```

Ask about shape: if those were connected, it would be a circle, right? That's how many degrees? 360. So if you wanted three circles, how many degrees would you need?

(Some student says 1080. "You violated our first rule – let the computer do the math!" So: 360×3 .)

You can have the students set their name(s) on their screen in the Virtual Proctor (if you are using it) - at virtualproctor-tkp-appspot.com by having them add the line below (passing in the names of the kids as a String to the set name method of the Virtual Proctor object).

```
1 VirtualProctor.setName("LynnAndSamantha");
```

Put up virtual proctor on screen and let them play with it, seeing what everyone's doing.

If you are planning to use the Spiral (optional) recipe, then you may want to add a second (nested) for loop in the variation for SimpleSquare, as the Spiral recipe includes this concept, including naming the local variables differently, i.e. 'i' for the first for loop and 'j' for the second for loop.

(Optional)

We have added the ability to set the name for your Tortoise. You can also get the current name (which is provided by default and can be updated using the 'setter'. An easy way to see the current value of the name variable is to use a message box.

```
1 Tortoise.setName("Ada Lovelace");
2 String myName = Tortoise.getName();
3 MessageBox.displayMessage(myName);
```

Part 3 – Quiz: SimpleSquareQuiz.java

To locate the .java file, use the keyboard shortcut "cmd-shift-T: SSQ" for (SimpleSquareQuiz) to bring up the Eclipse 'find file' menu

All TKP quizzes are written to let students celebrate their understanding of the concepts presented. Quizzes are written so that all students who have completed the previous recipes should be able to get 100% correct.

As a teacher, your role is to facilitate (but **not** to give the correct answers!) by guiding kids toward success. Here are some teaching tips for TKPJava quizzes:

- Kids work in pairs, teacher facilitates and helps get to kids 'unstuck' by asking questions.
- Tell kids that they simply enter code matching each recipe line, then run and the quiz will grade that line (those lines) to show whether they passed that line (and the entire test).
- Reminds kids to run after **each** question before deleting the English comment line.
- Don't show – suggest how they can discover for themselves by asking questions.

E.g. "What do the English say exactly here?"

"That method doesn't seem to be working for you, is there a different method you could try?"

."

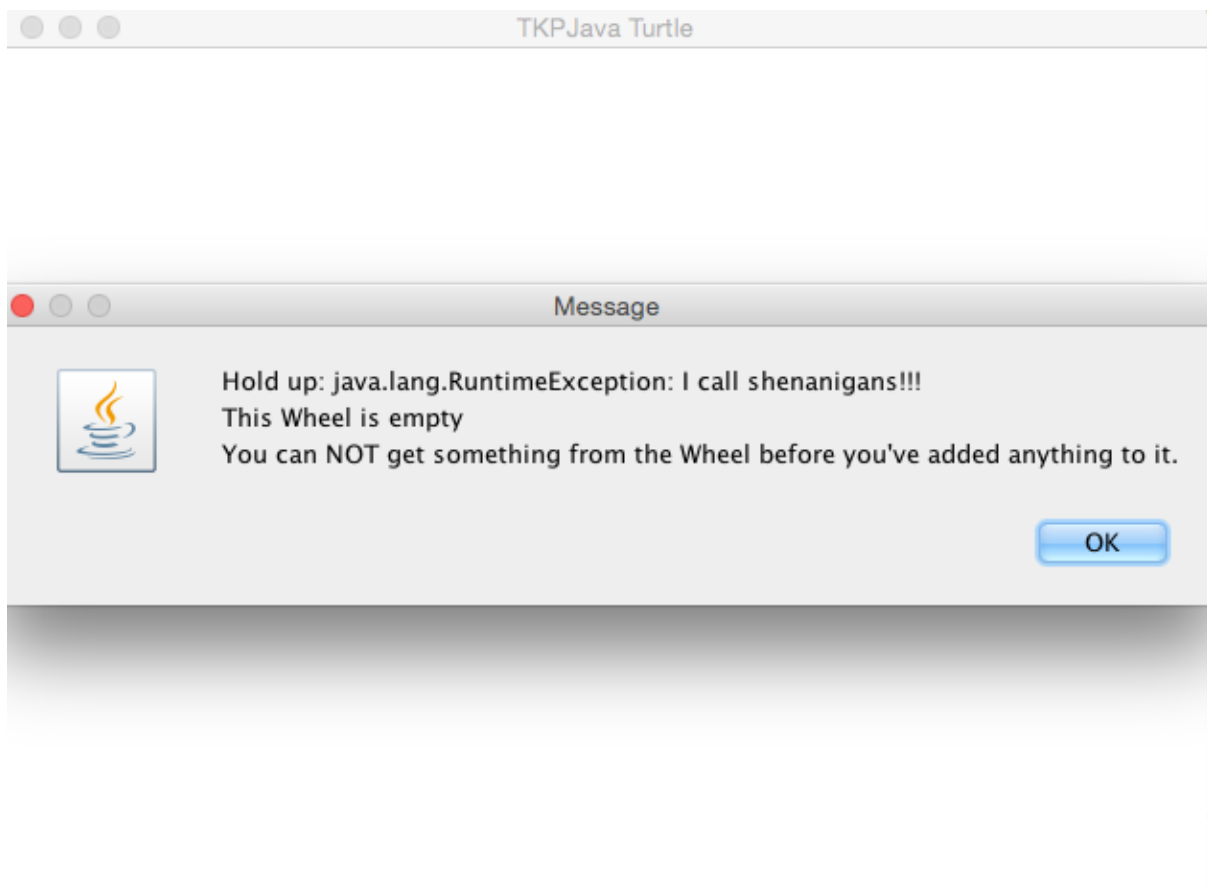
"You haven't gotten question 1 working. Let's undo and get that working before doing question 2."

Part 4 – Recipe: Spiral

Open “Spiral.java” - You should point out the ‘ColorWheel’ object, as this is the first introduction of this object.

You will introduce the idea of different loop variable names, i.e. ‘i’ and ‘j’ in nested for loops.

Additionally the idea of run-time exceptions is introduced by the deliberate execution path of using colors from the ColorWheel BEFORE colors are added. Having the kids code in the order listed will generate a custom exception in the console window, which is expected and provides an opportunity to introduce the ideas of execution order and null values in a fun way. We added the Java ‘try...catch...’ exception handling code to surface the expected runtime exception in a MessageBox (shown below) , in case you aren’t running Eclipse in debug mode.



Tip: Running Eclipse in debug allows you to see any generated exception messages in the Console window by default.

Here is a video recap of the recipe as well – <https://www.youtube.com/watch?v=k1EQTcjYKM>

If kids finish early, they can try variations. Here’s a video showing how to teach Spiral:

*

Part 5 - Additional Recipe: Make A Logo

This recipe continues introducing the idea of using an object *other than the Tortoise* when coding TKPJava. The new object in this recipe is the 'TKPLogo'. The TKPLogo object has a number of methods which demonstrate a couple of ideas:

- Useful, readable method names - ask the class 'what do think this method does **BEFORE** students run the code
- Reinforcement of tool usage - while method names are English-like, they are not exactly the same as English. Kids will code *much faster* if they following this pattern (and do NOT type out the object and method names letter-by-letter):
 - Type 'TK... , then ctrl+space, then click on 'TKPLogo' and press enter to auto-complete
 - Type a '.' (dot) after the TKPLogo, then ctrl+space
 - Select the method in the list, press enter to auto-complete it
- Introduction of the idea of abstraction. In a previous recipe (Spiral), kids coded the instantiation of the ColorWheel, i.e. ColorWheel.addColor(PenColors.Blues.Blue); etc... In this recipe they do not have to do that. Ask them why not, and also how this same result is achieved.

Additionally ask about exactly which sets of steps the TKPLogo.draw_tkp_T(); is performing, i.e. Tortoise.move(x); Tortoise.turn(x), etc...

You may want to draw the steps out on a whiteboard. You can also open the source code for the [TKPLogo object](#) and show the students the amount of abstraction included in this recipe.

*

Part 6 - Additional Recipe: Double Loop

This recipe introduces the idea of a nested loop and also of using different variable names for loop variables, i.e. 'i' and then 'j'. It also introduces the idea of 'hiding' the Turtle. It is designed as a starter to give kids more ideas for variations. Also via the double-loop they can visually understand 'for loops' more deeply.

Another concept introduced in the recipe is that of translating a recipe which includes mostly English comments, but also some existing Java code. The reason for this type of exercise is to allow for more complex recipes, but to still keep the number of steps the students complete relatively short. In addition the complexity of differentiating between English comments and Java code is supported using this partially completed' lesson.

*

Part 7 – Deep Dive: DeepDive01ForLoops.java

IMPORTANT Run each test as a first step. It should fail! Read the failure output.

Tip: To help kids understand 'how to see what you missed'

This one: <http://www.youtube.com/watch?v=vJG698U2Mvo>

Another one: http://www.youtube.com/watch?v=IGQmdoK_ZfY

Yet another one: [McGurk effect] (<http://youtu.be/G-IN8vWm3m0>)

Say: *'No matter what happens, you're going to miss a lot of it. So one way to counteract it is to get everyone to volunteer what they see, even if they think it's very obvious. (Could be as obvious as a gorilla walking across screen, since someone might not have seen it!).'*

SETUP

Chairs at front, circle/oval/whatever – computer at desk to the side.

Give kids post-its.

Each kid writes an observation and the last thing they just did. No talking – they just write. Then collect, and read them to the class.

PROCESS

When going through Deep Dive, one student will sit @ computer, another will be standing and will tell them what to type.

After each test, standing student gives observation (/explains why they entered what they did), then they rotate.

If they ask “Does it work if...” then try it and see!

SHOW THEM THE PROCESS

numbersDoNotNeedQuotes

Put cursor on method name. Then click on “numbersDoNotNeedQuotes”, run.

Show that it does not work.

Walk them through getting it to work, then have them write observations on it. They read.

defaultWidthForTheTortoise

Show failure trace – how do we find the right #?

stringsNeedQuotes

Type green

Did not work. Why? Look at the title!

Change to “green”.

We done? Nope! Must run.

Double click on the failure trace when running the failed program – show the window that pops up.

theTortoiseTurns15Twice

How is this different from the last one? (Turning twice, but looking at the angle which results from both turns)

combiningNumbers

How is this different from the last one? (Adding numbers, looking for total)

combiningText

How is this different from the last one? (Adding strings just combines them one after the other.)

combiningTextAndNumbers

How is this different from the last one? (Adding a number to a string treats the number as a string.)
Useful techniques – asking how one is different from the last. (E.g., “assigningVariables” vs. “how-FastCanTheTortoiseGo”)

combiningTextInALoop

Ensure the kids get what happened here. Maybe pick one from audience to explain.

forLoopsEndAtTheLine

A fairly advanced thing
Double-click in blue thing. Debug dot.
Now, right click on for loop in left part... sends you to debugger.
Use “step over” (yellow arrow top right) to see individual steps. Show the kids how this works.

forLoopsCanStartAnywhere

Same as above – have one explain

forLoopsCanSkip

Same as above – have one explain
*

Part 7 – Worksheet: To locate ‘1_SimpleSquareWorksheet.docx’ for printing

Download, unzip and print from - [here](#) (contains all worksheets for all courses)
Have kids circle part of code, part of English, draw line between them.
Have them go through it – use teacher version for notes.
*

3 Teaching TKPJava Course 02 - Methods & Variables

Using Extract Variable and Extract Method

Preparing to Teach this Course

Every Course

:hourglass: **Install** the [TKPJava courseware](#)

:green_book: **Read** this lesson plan page

:computer: **Code** all recipes yourself

:bulb: **Review** the [TKPJava Language pptx](#)

:fax: **Print** the [keyboard shortcut sheet] (<http://www.slideshare.net/lynnlangit/tkpjava-eclipse-and-codenvy-ide-key>)

:fax: **Print** the main [recipe worksheet](#)

:swimmer:

Part 1 - Recipe: Houses

Recipe Concepts

Create methods and run methods. The programmer way to say run a method is by saying 'call' a method.

Recipe Guidelines

We are introducing the idea of creating a variable (via refactoring) and also of setting the value of that variable (multiple times) during this recipe. We find it helpful to use the word "current" when talking about variables, i.e. "What is **current** value of the height", etc...

Another teaching tip is to start by assigning a number to the height BEFORE creating the variable. If you teach this way, remember to have the kids run the recipe each time as you lead them from a number, to a variable name and eventually to a variable which has the current value of the number, i.e. 40.

Also, lead the students to NOT delete the English comment until AFTER they've completely translated the number into a working variable. This is an important programming concept, we call it 'fake it until you make it'.

To Refactor: refactor the fake and extract it as a variable, name it "height".

We have now created line 16: `int height=1000`

But it is not translating line 15, so move the code to follow the English its translating and replace the fake number with the information we learn: 40

```
1 1.1: Tortoise.move(height);
2 1.2: int height = 40;
```

Delete English lines 1.1 and 1.2

This one's already done now! Run to test, then English can be deleted.

[For 2-8, do them quickly as a class. If they forget to turn numbers negative, just run it and let them be confused.]

```
1 2. Tortoise.turn(90);
2 3. Tortoise.move(30);
3 4. Tortoise.turn(90);
```

Ensure they do this the lazy way! (And the following)

```
1 5. Tortoise.move(height);
```

Make sure kids are using "height", not "40"

```
1 6. Tortoise.turn(-90);
2 7. Tortoise.move(20);
3 8. Tortoise.turn(-90);
4 9: sub-recipe
```

THIS IS WHERE WE GET TO SOMETHING TOTALLY NEW.

Anyone notice something different about this recipe? – It's like a recipe within a recipe!

You can show a short (less than 3 minute) video here to show how to do this. [How to extract a method](#)

This is a sub-recipe. When you make a cake, there is the recipe for the whole cake and the recipes for the parts: cake, filling, frosting;

Show them how to extract a method. Highlight from beginning of method to end, INCLUDING comments for line 9 (top and bottom). Choose Refactor Extract Method, then use name of sub-recipe: drawHouse.

Delete all the line 9s, run and see that it does exactly the same thing.

When we refactor, code should run exactly the same.

Have kids look at it and try to figure out – what did we just do? Why might this be useful?

Give terminology: method. Why would we do it? If we're doing the same sort of thing over and