1. All code is in Github repository
    a. https://github.com/in28minutes/JavaTutorialForBeginners/blob/master/Level1.md
2. How do we make use of the JUnit Tests in Exercises?
3. Add some eclipse magic throughout - in every video!!!
4. Exercises
    a. Make this program compile!
    b. Make the logic correct!
5. Idea is to cover all the basic topics first
    a. 5 examples with the new approach are in the GIT repository
    b. Rest of the examples need to be organized topic wise
    c. We would want to create a multi level java course
        i. Initial thinking was - needs more thought put in...
            1. Basics
            2. Level 1
                a.
            3. Level 2
                a. Advanced OOPS
            4. Level 3
                a. Collections and Generics
            5. New Features
6. Level 1
    a. Object Class State and Behavior : What is Object? What is Class? What is state of an object? How do we represent state? (using variables.) What is behavior of an object?
        i. What is Class? What is Object?
            1. What is special about the main method?
            2. Object
                a. Type or Class
                b. Made up of other objects
                c. Has an interface
                    i. Defines what messages it can receive
            3. Object Oriented Program is a bunch of objects sending messages to each other.
            4. Examples
                a. Switch
                    i. On or Off
                b. Account
                    i. Withdraw
                    ii. Deposit
                    iii. changeAddress
                    iv. get Address
                    v. getName
                    vi. getBalance

            vii. printStatement

ii. State of an Object
    1. What is a variable?
        a. What is variable, declaration,initialization,expressions
        b. Variable declaration int balance
        c. box with name,type and value.
        d. int is data type - +1, 0, -1. diff types:float - decimal 0.1 0.22
        e. char 'a' 'b'. more about these in next classes
        f. variable has name, type & value. Value can change. one time 5, later 10
        g. declaration int score; declaration tells type & name
        h. initialization - first value of variable. run twice with diff values 0,10.

iii. What is assignment?
    1. Basic Operations
    2. Debug

iv. Methods. Behavior
    1. Typical Mistakes a programmer can make

v. Create more state and behaviour. Print the state of an object.

vi. Why is Java Popular? What is Platform Independence?
    1. Bytecode, JDK, JRE, JVM

vii. Constructor

viii. Let's Break This
    1. Most important part of programming is debugging.. Finding out what is wrong. Let's take first steps towards learning debugging now! Let's have some fun as well. Try removing any character or line and try and guess how the program behaves.

ix. Parameters to a method -> Increase speed by 20

x. Football Scorer Approach 1 -> team1score, team2score

xi. Football Scorer Approach 2 -> Team team1; Team team2;
    1. Difference between approaches
    2. Approach 2 is more extensible
    3. How we organize objects is very important!

xii. Return Value
    1. Find Largest Number
    2. Which team won? Add a method to Football Scorer Approach 1
    3. Exercise: Create a small program to find area of Rectangle.
    4. Exercise: is odd or is even

xiii. Exercise
    1. 24 hour timer class with a tick function
    2. Bank Account

xiv. Types of Variables - An introduction
    1. Primitive Data Types need not be created using new operator. That is the main difference

        2. byte cannot store even one day cricket match score. might be good to store football score
        3. You cannot store fractions into integer variables
        4. Float and Double are not precise. Avoid for financial calculations.
        5. Default value is assigned to primitive variables by default. Use Scorer class to show it for integer.
        6. Player class
            a. With a name. We want to assign a player name. String name.
                i. Add names to Sachin and Ronaldo and explain
            b. Remove name for ponting and see what happens.
            c. Assigning default value to a member variable.
            d. Creating getters and setters
                i. Method Parameters
        7. A number of exercises!
            a. isVowel
            b. calculateGrade
        8. Local vs member variables

b. Quick Intro To if, arrays, for, while and do while
    i. Exercise : If condition to enable maximum and minimum speed and gears limit. One of these can be a exercise
    ii. Why do we need an Array?
        1. Find sum of an Array
        2. Find max in an array
        3. Write this with for, while and do while

c. Programming Logic
    i. For Loop
        1. Exercise : Sum of Digits
        2. Exercise : Reverse a Number
        3. Exercise : Palindrome
    ii. Exercises
        1. Print array backwards
        2. Diving competition scoring
            a. Drop highest and smallest scores from 8 Judges and calculate average
        3. Factorial
    iii. Programming Logic : Arrays, Conditionals and Loops
        1. Arrays
        2. Section 8: Conditionals and Loops
            a. == vs =
        3. Simple Puzzles on conditions and loops
        4. Simple programs to improve programming logic

            a. If Statement - IfStatementTest
            b. Switch Statement - SwitchStatementTest
            c. DoWhileLoopTest
            d. WhileLoopTest
            e. ForLoopTest
            f. Enhanced For Loop Test
            g. Prime number
            h. Print Prime numbers below 100 using for(;;), while, do while

d. Java Built-in Utility Classes
    i. Wrapper Classes
        1. Creation - valueOf vs Constructor
            a. Look at the code of Integer.valueOf
        2. Reasons why we need Wrapper Classes
            a. • null is a possible value
            b. • use it in a Collection
            c. • Methods that support Object like creation from other types.. like String Integernumber2=newInteger("55");//String
        3. Autoboxing
            a. Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes.Auto Boxing helps in saving memory by reusing already created Wrapper objects. Auto Boxing uses the static valueOf methods.
    ii. String, StringBuffer and StringBuilder
        1. Performance : Avoid String Concatenation!
        2. Program : Find if String is a Palindrome
        3. Program : Code a String Splitter!
    iii. ArrayList
    iv. HashMap
    v. Exercise : Write a program to use some other built-in Java Class.
        1. Lets use the documentation which is available to do that!
e. Quick Introduction to Exceptions
    i. What is an Exception?
        1. Change Gears less than 0
    ii. Exercise : BankAccount
        1. withdraw
            a. Exception
        2. deposit
            a. Exception
        3. printStatement

            4. getBalance
    f. A few Good Habits before going to Level 2
        i. Comments & Readable Code
        ii. JavaDoc
        iii. Static Imports
        iv. Coding Guidelines & Standards
        v. Unit Tests
            1. Array Tips and Tricks
                a. arrayoutofbounds
                b. default values in array - new int[5];
                c. Arrays.fill
                d. Loop an array
                e. Sort an array
                    i. assertArrayEquals

7. Level 2
    a. Programs using Java Utility Classes
        i. Performance Impact of String StringBuffer and StringBuilder.
            1. Write a JUnit to test this!
        ii. HashMap : distinct chars in String
        iii. Distinct strings in an array
        iv. No of occurrences of each character in String
        v. Create an Address Book
    b. Basics of Object Oriented Programming
        i. Interface
            1. An interface defines a contract for responsibilities (methods) of a class.
            2. An interface is a contract: the guy writing the interface says, "hey, I accept things looking that way"
            3. Interface represents common actions between Multiple Classes.
            4. What does this mean : “Always code to an interface!”
            5. Examples
                a. Video game console :
                    i. Exposes the interfaces for the buttons
                        1. Left, right, up, down, green, red, blue
                    ii. Game Writer
                        1. Provide implementation for this interface
                b. Two Companies are working on a complex project. Let’s take a simple example. One company is responsible for sorting.
                    i. First they agree on an interface.
                    ii. inferface Sortable { public List<String> sort(List<String> items) }

iii. Company responsible for sorting will work on implementing the interface.
iv. Company writing the application will use a dummy implementation of the interface.

c. Example in Java api : Map interface, Collection interface.

6. Tips and Tricks
    a. Variables in an interface are always public, static, final. Variables in an interface cannot be declared private.
    b. Interface methods are by default public and abstract. Before Java 8, A concrete method (fully defined method) cannot be created in an interface.
    c. An interface can extend another interface.An interface cannot extend a class.
    d. A class can implement multiple interfaces.
    e. An example of a class in the JDK that implements several interfaces is HashMap, which implements the interfaces Serializable, Cloneable, and Map. By reading this list of interfaces, you can infer that an instance of HashMap (regardless of the developer or company who implemented the class) can be cloned, is serializable (which means that it can be converted into a byte stream; see the section Serializable Objects), and has the functionality of a map.

ii. Inheritance

iii. Abstract Class

1. Placeholders for actual implementation
2. Abstract class defines behaviour, Subclasses implement the behavior.
3.
```
public abstract class Instruction { void perform() {
firstStep(); secondStep(); thirdStep(); } abstract void
firstStep(); abstract void secondStep(); abstract void
thirdStep(); }
```
4. An abstract class is a class that cannot be instantiated, but must be inherited from. An abstract class may be fully implemented, but is more usually partially implemented or not implemented at all, thereby encapsulating common functionality for inherited classes.
5. Template Method Pattern
    a. ensureSecurity()
    b. if(validate())
        i. executeRequest()
6. An example of an abstract class in the JDK is AbstractMap, which is part of the Collections Framework. Its subclasses (which include HashMap, TreeMap, and ConcurrentHashMap) share many

methods (including get, put, isEmpty, containsKey, and containsValue) that AbstractMap defines.

7. Example abstract method : public abstract Set> entrySet();
8. Another Example - Spring AbstractController

iv. Method overloading
    1. Constructors
        a. public HashMap(int initialCapacity, float loadFactor)
        b. public HashMap() {
        c. public HashMap(int initialCapacity)
    2. Methods
        a. public boolean addAll(Collection<? extends E> c)
        b. public boolean addAll(int index, Collection<? extends E> c)

v. Method overriding
    1. HashMap public int size() overrides AbstractMap public int size()

vi. Interface vs Abstract Class
    1. Account (SavingsAccount, CheckingAccount, SeniorCitizenSavingAccount)
    2. Flyable (Bird, Aeroplane)

vii. Object - equals and hashCode methods

viii. Constructors
    1. Default Constructor is the constructor that is provided by the compiler. It has no arguments.
    2. A constructor can call the constructor of a super class using the super() method call. Only constraint is that it should be the first statement.
    3. Another constructor in the same class can be invoked from a constructor, using this({parameters}) method call.
    4. If a super class constructor is not explicitly called from a sub class constructor, super class (no argument) constructor is automatically invoked (as first line) from a sub class constructor.

c. Basics Collections
    i. Collection Interface Hierarchy
    ii. Collection & List Interface methods and classes - ArrayList, Vector & LinkedList
    iii. ArrayList
        1. List of Questions
        2. List of Answers
        3. Match Questions and Answers and decide rating!
    iv. Set interfaces and implementations - HashSet, LinkedHashSet and TreeSet
    v. Map interfaces and implementations - HashMap, LinkedHashMap and TreeMap
        1. HashMap : Number of occurrences of each character in a String.

            2. Program : Find unique characters in a String
    d. Generics
        i. Generics
    e. Modifiers
        i. Access Modifiers - public, private, protected and default
        ii. Final method, variable and class
        iii. Static variables and methods
    f. Enum
        i. Enum Colors, grades, sports
    g. Variable Arguments
    h. Utils
        i. BigDecimal
        ii. Date
        iii. Calendar

8. Level 3
    a. More Object Oriented Programming
        i. What is Abstraction?
            1. Computers can only understand 0 or 1. Low level assembly language was used earlier. High Level Languages like java are an abstraction on top of these.
        ii. What is Coupling?
        iii. What is Cohesion?
            1. Example : MVC pattern. Each part has individual responsibility
            2. Example : Layering of applications. Each layer has individual responsibility.
            3. Example : How Spring Framework is organized into modules!
        iv. What is Encapsulation?
            1. Protect your classes from unnecessary changes!
            2. Think about driving a car. We just change gears and move the driving wheel. We are not aware of how the internal mechanics work. This is a good Example of encapsulation.
                a. ChangeGears(), moveDrivingWheel()
            3. All manufacturers offer same interface, even though the internal details of how the car works are different.
            4. Switch ON and OFF
    b. Inner Classes
        i. Inner Class and Static Inner Class
        ii. Static and Member Initializers
    c. Exception Handling & Logging
        i. Exception Handling is an example of Chain of Responsibility Pattern.
        ii. Exception Handling - try, catch and finally
        iii. Checked and Unchecked Exceptions
        iv. Throwing an Exception

            v. Creating Custom Exceptions
    d. Advanced Collections
        i. Queue interfaces and implementations - Deque and BlockingQueue
        ii. Concurrent Collections
            1. Concurrent Collections - CopyOnWriteArrayList
            2. CompareAndSwap, Locks and AtomicOperations
    e. Multithreading
        i. MultiThreading - Need for Threads and Creating Threads
        ii. Thread states, priority, ExecutorService and Callable
        iii. Synchronization of Threads. join, wait, notify and notifyAll methods
    f. Functional Programming - Lamdba Expressions and Streams
        i. Functional Programming Examples - Streams and Lambda Expressions
        ii. Functional Programming Questions and Answers
    g. More Good Practices
        i. Advanced Refactoring?????????

9. Level 4
    a. Good Design???
    b. TDD
    c. Maven - 10 Tips
    d. JUnit - 10 Tips
    e. Code Quality & Standards - 10 Tips
        i. 4 Principles of Simple Design
        ii. Effective Java
    f. Refactoring - 10 Tips
    g. Eclipse - 10 Tips
    h. Java - 10 Tips
        i. Do not use float or double for monetary calculations
    i. Design Patterns - 5 Examples
    j. SOLID PRINCIPLES
    k. How do you continue you journey?

Rest of the Topics Organization

- Miscellaneous
    - What happens in the background when we run the program?
    - Quick Revision of all that we learnt until now!
    - Platform Independence & JVM vs JRE vs JDK
    - How is stuff stored in memory? Pass by reference vs Pass by value
    - Java Classloaders
    - Garbage Collection
    - Serialization
    - What is an anonymous class?
    - Boy Scout Rule

What do you think about while you code

- Am I going to understand this in 3 months time?
- Am I trying to be too clever: is there a simpler way to get the job done?
- Can I write this in such a way as to make some parts re-usable?
- Have I written anything before that I could re-use here?
- What's for dinner?
- Am I going to be able to easily test this?
- Is the first programmer who will read the code going to send a snippet to The Daily WTF?

List

---

http://www.youtube.com/watch?v=Y8bY0jxk3LE
http://www.youtube.com/watch?v=fefVIUvfFB4
http://www.youtube.com/watch?v=c8ZQwz76wuM
http://www.infoq.com/presentations/10-Ways-to-Better-Code-Neal-Ford
http://nealford.com/downloads/JAX_Keynote_2009%28Neal_Ford%29.mp4
http://www.infoq.com/presentations/craftmanship-ethics
http://www.infoq.com/presentations/Robert-C.-Martin-Bad-Code
http://www.infoq.com/interviews/coplien-martin-tdd
http://www.infoq.com/presentations/principles-agile-oo-design
http://www.youtube.com/watch?v=mslMLp5bQD0
http://www.viddler.com/explore/sergiopereira/videos/7/
http://www.viddler.com/explore/oredev/videos/15/
http://www.parleys.com/#st=5&id=2168&sl=14
http://www.parleys.com/#st=5&id=1491&sl=1

Books

List

- Code Complete by Steve McConnell.

- Clean Code- A Handbook of Agile Software Craftsmanship
- Working Effectively with Legacy Code by Michael Feathers
- The Pragmatic Programmer: From Journeyman to Master by Andrew Hunt
- Patterns of Enterprise Application Architecture by Martin Fowler
- Effective Java (2nd Edition) by Joshua Bloch
- Extreme Programming Explained: Embrace Change (2nd Edition) by Kent Beck
- Agile Software Development, Principles, Patterns, and Practices by Robert C. Martin
- Refactoring: Improving the Design of Existing Code by Martin Fowler
- Test Driven Development: By Example by Kent Beck
- Code Craft
- The Productive Programmer
- Disciplined Agile Delivery
- Anti Patterns

Katas/Examples

Refactoring Examples :http://refactormycode.com/codes/recent/java
Roman Numerals:http://vimeo.com/33841375
http://ubuntuforums.org/showthread.php?t=1714324
http://www.docondev.com/2011/12/roman-numeral-kata.html
http://www.codingdojo.org/cgi-bin/wiki.pl?KataCatalogue
http://schuchert.wikispaces.com/Katas
http://www.butunclebob.com/ArticleS.UncleBob.ThePrimeFactorsKata
http://sites.google.com/site/tddproblems/all-problems-1
http://butunclebob.com/ArticleS.UncleBob.TheBowlingGameKata
http://www.docondev.com/2011/01/sharpening-saw.html
http://codekata.pragprog.com/2007/01/code_kata_one_s.html
http://programmingpraxis.com/contents/chron/
http://www.codinghorror.com/blog/2008/06/the-ultimate-code-kata.html
http://www.knowing.net/index.php/2006/06/16/15-exercises-to-know-a-programming-language-p
art-1/
http://vimeo.com/user3159463/videos/sort:plays
http://johannesbrodwall.com/2010/04/06/why-tdd-makes-a-lot-of-sense-for-sudoko/
http://www.viddler.com/explore/GreggPollack/videos/29 LCD Numbers Kata by Corey
http://osherove.com/tdd-kata-1/

http://osherove.com/tdd-kata-2/

Miscellaneous

http://norvig.com/21-days.html Teach Yourself Programming in Ten Years
http://programmer.97things.oreilly.com/wiki/index.php/Contributions_Appearing_in_the_Book
http://programmer.97things.oreilly.com/wiki/index.php/Other_Edited_Contributions
http://samizdat.mines.edu/howto/HowToBeAProgrammer.html
http://www.javacodegeeks.com/2011/07/top-97-things-every-programmer-or.html
http://developerart.com/publications/4/wise-programming-techniques-for-writing-quality-code

The Joel Test: 12 Steps to Better Code -
http://www.joelonsoftware.com/articles/fog0000000043.html
http://www.javacodegeeks.com/2010/12/things-every-programmer-should-know.html
http://www.indiangeek.net/wp-content/uploads/Programmer%20competency%20matrix.htm

Code Smells

http://sourcemaking.com/refactoring/bad-smells-in-code
http://www.codinghorror.com/blog/2006/05/code-smells.html
http://www.soberit.hut.fi/mmantyla/BadCodeSmellsTaxonomy.htm

Anti Patterns

http://sourcemaking.com/antipatterns
http://en.wikipedia.org/wiki/Anti-pattern

TDD

http://programmers.stackexchange.com/questions/86636/tdd-vs-productivity
http://osherove.com/blog/2005/4/3/naming-standards-for-unit-tests.html
http://vimeo.com/5812605
http://vimeo.com/5813145
http://xunitpatterns.com/

http://stackoverflow.com/questions/96297/what-are-some-popular-naming-conventions-for-unit-tests
http://blog.james-carr.org/2006/11/03/tdd-anti-patterns/
http://www.infoq.com/interviews/coplien-martin-tdd
http://powersoftwo.agileinstitute.com/2011/11/five-key-ingredients-of-essential-test.html
Move Specificity Towards the Tests http://vimeo.com/5895145

Refactoring

Code Examples : http://refactormycode.com/codes/recent/java
Getting Empirical about Refactoring
http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=16679&tth=DYN&tt=siteemail&iDyn=2
http://sourcemaking.com/refactoring
http://www.cleal.com/refactoringgolf.html
http://addcasts.com/2011/12/12/episode-15-michael-feathers-chats-about-refactoring-in-the-fourth-dimension-seducing-end-users-and-letting-code-die/
SOLID
http://hanselminutes.com/145/solid-principles-with-uncle-bob-robert-c-martin
41st episode of the StackOverflow podcast, where Joel and Jeff sit down with Robert Martin aka “Uncle Bob”, and discuss software quality, the value of software engineering principles, and test-driven development.
http://blog.sanaulla.info/2011/11/16/solid-single-responsibility-principle/#more-1134
http://blog.stackoverflow.com/2009/02/podcast-41/
http://www.objectmentor.com/resources/articles/ocp.pdf
http://www.itmaybeahack.com/book/oodesign-java-2.1/latex/BuildingSkillsinOODesign.pdf