Maharishi International University 1971-1995

MAHARISHI UNIVERSITY OF MANAGEMENT

Professional Excellence and Higher Consciousness

CS 401:
Modern Programming Practices

Dr. Paul Corazza

2017

Maharishi's Tenth Year of Global Raam Raj

Maharishi University of Management is an Equal Opportunity Institution.

CS-401 Modern Programming Practices

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
OOAD	**AM: Lesson 1:** *The OO Paradigm for Building Software Solutions* **PM: Lab 1**	**AM: Lesson 2:** *Associations among Objects and Classes* [Lab 1 due 10 AM] **PM: Lab 1 solutions, Lab 2**	**AM: Lesson 3:** *Inheritance and Composition* [Lab 2 due 10 AM] **PM: Lab 2 solutions, Lab 3**	**AM: Lesson 4**: *Interaction Diagrams* [Lab 3 due 10 AM] **PM: Lab 3 solutions, Lab 4**	**AM: Lesson 5**: *Inheritance and Abstraction* [Lab 4 due 10 AM] **PM: Lab 4 solutions, Lab 5**	**AM: Lesson 6, Part I**: *Introduction to JavaFX* [Lab 5 due 10 AM]
	AM: Lesson 6, Part II: *JavaFX and SceneBuilder* **PM:** Design Workshop and Project	**AM:** Project **PM:** Project	**AM:** Project **PM:** Project	**AM***:* Review points for Midterm **PM:** Project/Study for Midterm	**AM :** Review points for Midterm. *Late morning*: Early-bird project presentations **PM:** Review for exam	**MIDTERM EXAM**
OOP	**AM: Lesson 7**: *Interfaces in Java 8 and the Object Superclass* [Lab 6 due 10 AM] [Submit Projects] **PM: Lab 5 solutions, Lab 7**	**AM: Lesson 8**: *Functional Programming in Java* [Lab 7 due 10 AM] **PM: Lab 6, 7 solutions, Lab 7**	**AM: Lesson 9**: *The Stream API* [Lab 8 due 10 AM] **PM: Lab 8 solutions, Part of Lab 9**	**AM: Lesson 9**: (continued) **PM: Some Lab 9 solutions, the rest of Lab 9**	**AM: Lesson 10** *Best Programming Practices with Java 8* [Lab 9 due 10 AM] **PM: Lab 9 solutions, Lab 10**	**AM:** [Lab 10 due 10 AM] Lambda/Stream practice exercises **Lab 10 solutions**
	AM: Lesson 11 *Generic Programming* **PM: Lab 11**	**AM & PM:** [Lab 11 due 10 AM] Final exam review points **Lab 11 solutions**	**AM**: Review for Final exam **PM:** Study for final	**10-12:30 FINAL EXAM** **2-4 PM PROGRAMMING TEST**		

Modern Programming Practices

Programming is the most basic part of Computer Science, as it is the basic language for expressing structures, processes, algorithms, systems, everything computable and computing related.

There are many different approaches and languages for programming, but currently object-oriented is the dominant model, and Java is the dominant language.

This course will provide an introduction to the OO paradigm, including analysis and design; to the use of UML to develop and communicate requirements and designs; and to implementation techniques, including an introduction to advanced features of the Java programming language.

Topics include:

- Objects and classes
- Analysis of requirements and development of design
- UML class, sequence, and object diagrams
- Relationship between UML diagrams and Java implementation
- Inheritance, interfaces and polymorphism
- Open-closed principle
- Collection processing with lambdas and streams
- Advanced unit-testing and exception-handling
- Annotations
- Programming with multiple threads
- Java generics and generic programming

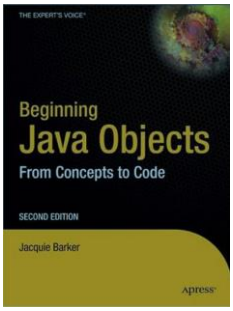
Core Outcomes. By the conclusion of MPP, the student will be able to demonstrate knowledge and skill in the following areas:

1. Ability to create use case, class, sequence, and object diagrams based on a detailed problem statement
2. Ability to translate UML diagrams into a full Java program
3. Ability to work with lambda expressions, method expressions, and their representation as nested classes
4. Ability to implement SQL-like queries on Collections in the form of Stream pipelines, making use of Java's rich Stream libraries
5. Beginning-level knowledge of JavaFX, Java generics, and generic programming
6. Understand the important role of the inward stroke of programming in the field of software development

Outcomes 1-5 will be the technical focus of the course; they will be addressed in lectures, in-class exercises, labs, exams, and in the course project. Outcome 6 will mainly be addressed experientially, by group meditation practice, but also through discussion of main points for each lesson.

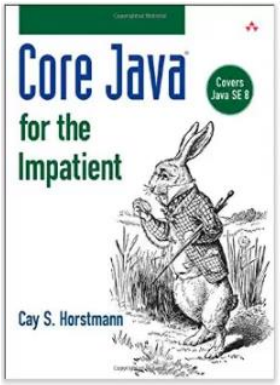
Required Course Text

- Jacquie Barker, *Beginning Java Objects: From Concepts to Code*, 2nd edition, Apress, 2005. Intro to OOAD.

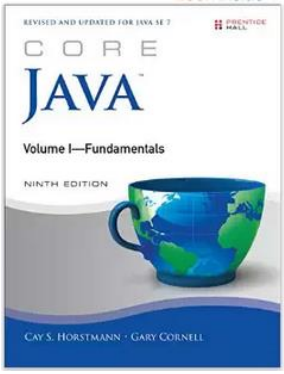


Additional Books and Resources (not required)

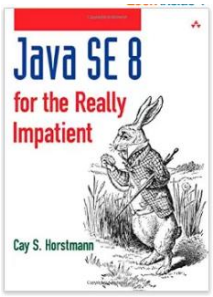
- Cay Horstmann, ***Core Java for the Impatient***, Addison-Wesley, 2015 (recommended) Covers Java 8



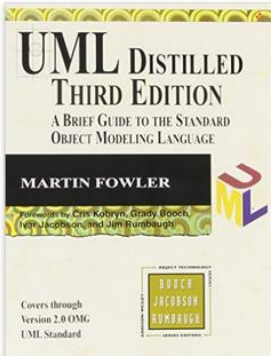
- Cay Horstmann**,** *Core Java Volume I--Fundamentals (9th Edition)*, Prentice-Hall, 2012. Covers Java 7, treatment is more thorough.



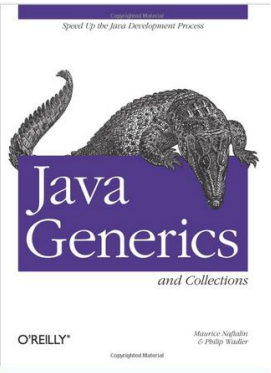
- Cay Horstmann, *Java SE 8 for the Really Impatient*, Addison-Wesley Professional, 2014. Covers JavaFX quickly.



- Martin Fowler, *UML Distilled, 3rd Edition,* UML syntax and best practices.



- Naftalin and Wadler, *Java Generics.* Introduction to Java generics.



- J. Bloch, *Effective Java,2nd Edition.* Best practices in Java design and implementation.

In-Class Exercises

During each lecture, a few exercises related to the lecture material will be mentioned. You will have 7-10 minutes to work on a given exercise, either individually or with one or two other students. These exercises will help you master the concepts of the course more quickly. All these exercises are stored in an Eclipse project named InClassExercises, which can (and should) be downloaded from Sakai onto your laptop. Some exercises are in the form of code; others are PDFs or JPGs; but all can be found in this project. When the time comes each day for an in-class exercise, be ready to read instructions and work with startup code located in this project. You will not submit your work on these, though some of the exercises are continued in the lab for that lesson.

Labs and Presentation

Each lesson has a corresponding lab or "assignment". Some labs require Java code; others require creation of UML diagrams and other forms of documentation. Your work for each lab **should be submitted as a group**.

Groups may consist of 2-3 persons. Group submissions are done through Sakai (see next section for details).

During the second week of class, there will be a group project. Each group will submit their project on the Monday of the following week (instructions for submission will be announced). There will be an opportunity for a few groups to do project presentations for extra credit on the last Friday of the second week

Submitting Your Labs

Instructions for each lab will be shown in a Sakai assignment ("Assignments" is an option in the left panel of your Sakai app). Each assignment will provide instructions and, if necessary, other materials. When you are ready to turn it in, please follow these instructions:

1. An assignment is submitted by attaching items (codes, jpeg's, other documents) to the assignment page (you will see instructions for how to do this in the assignemnt itself).
2. *One person* in your group will submit the work for your entire group. (Everyone in your group will get the same grade for this submitted work. There is no need for more than one person to submit an assignment.)
3. There are several types of things you may need to submit. Put everything in a folder and zip it up; the name of the submitted file should be your group name + lab number. Example: Group5_lab3.zip
    a. When you submit code, include the src folder (which should just contain source code) – do not submit the entire project
    b. When you submit UML diagrams, submit only JPEG images of your diagrams, not the entire Star UML project.
    c. When you submit other documents, submit them in PDF format

Professional Etiquette

You are expected to dress and behave as you would in the context of a real IT job in the US. With Compro students, there are rarely any issues about professional etiquette, but you should be aware that you can lose up to 3 points (out of 100 total for the course) for failure to observe acceptable codes of dress and behavior. Be alert to the following:

1. Dress appropriately
2. Attend class in the expected way
3. Don't be late to class
4. Don't leave class before it is time to leave
5. Don't skip class
6. Respectful attitude toward the professor. (Arguing about grades is a dangerous path to walk.)

End-of-Block Programming Test

On the afternoon of the last Thursday of the block, you will be given a 2-hour standardized programming test, which will test your skills in two of the primary areas covered in the course: Use of Java 8 tools to process collections, and ability to convert a UML diagram into Java code. In order to pass MPP, you will need to pass this test. If you pass, your final course grade will be determined by the Evaluation Criteria shown below. If you do not pass, your final course grade will be C+ (or lower) and you will need to repeat (and pass) MPP in order to move on to other courses in the program.

Evaluation Criteria

The course grade will be determined according to the following (assuming that you pass the standard programming test given at the end of the block):

Activity	Percent Value
Labs	10%
Project	10% (12% e.b.)
Midterm Exam	40%
Final Exam	40%
Professional Etiquette	(worst: -3%)

Academic Honesty

Students are expected to submit only their own work (except for labs or other activities designated as group activities). Duing exams, they must not look at other students' work, discuss exam contents with other students at any time (including bathroom breaks), or attempt to access outside resources (such as internet or email). The academic dishonesty policy stated on the Compro website is reproduced here:

> **Academic Dishonesty:** Graduate students caught cheating will receive a grade of NC. A second case of cheating results in suspension from the university. Cheating includes copying from someone else as well as letting someone else copy your materials, or not following the policies during the test (e.g., not using a cell phone at any time; not having notes, etc)

Grading Scale

The following grading scale will be used in this course:

Range	Letter Grade
93 -100	A
90 - 92	A-
87 - 89	B+
83 - 86	B
80 - 82	B-
77 - 79	C+
73 - 76	C
70 - 72	C-
Below 70	NC

Lesson 1

The OO Paradigm for Building Software Solutions

Unlocking the Blueprint of Creation

Wholeness of the Lesson

In the OO paradigm of programming, execution of a program involves objects interacting with objects. *Analysis* is the process of understanding user requirements and discovering which objects are involved in the problem domain. *Design* turns these discovered objects into a web of *classes* from which a fully functioning software system is built. Each object has a type, which is embodied in a Java *class.* The intelligence underlying the functioning of any software object resides in its underlying class, which is the silent basis for the dynamic behavior of objects. Likewise, pure consciousness is the silent level of intelligence that underlies all expressions of intelligence in the form of thoughts and actions in life.

Main Point 1. Software is by nature complex, and the only way to manage this complexity is through *abstraction*.

Abstraction is at work when we discover the objects in the problem domain during analysis, and work with these to build a system during desing. Abstraction is also at work in creating maps of our objects in the form of UML diagrams.

In a similar way, to manage the complexities of life itself the technique is to saturate awareness with its more abstract levels so that all the details of any situtaion are appreciated from the broadest perspective. The abstract levels of awareness are experienced in the processs of transcending.

Main Point 2. The OO approach to building software solutions is to represent objects and behavior in the problem domain with software objects and behavior. One of the first steps in this process is to *locate* the objects implicit in the problem statement, and this is done by examining *nouns* and *noun phrases* in the problem statement. These words and phrases link the real world situation to the abstract realm of software objects. Likewise, linking individual awareness to its abstract foundation in fully expanded awareness is the basis for creating solutions to the real-world challenges of life.

Main Point 3. During OO Design, we specify in more detail the structure of classes. A class encapsulates *data,* stored as attributes and *behavior*, represented by operations. These are the static and dynamic aspects of any class, and a UML diagram for a class provides compartments for each of these.

These two aspects of a class – data and behavior – are aspects of anything the we encounter in creation. They give expression to the reality that life, at its basis, is a field of *existence* and *intelligence.*

CONNECTING THE PARTS OF KNOWLEDGE
WITH THE WHOLENESS OF KNOWLEDGE

CLASS DIAGRAMS

1. Class diagrams display the data and behaviors of a class

2. Class diagrams provide an (abstract) representation of a specific real word problem domain

---

3. *Transcendental Consciousness* is the simplest state of awareness, where the mind goes beyond thoughts and concepts to the most abstract level of awareness.

4. *Impulses Within the Transcendental Field.* The hidden self-referral dynamics within the field of pure intelligence, on the ground of pure silence, provide the bluprint for the perfectly orderly unfoldment of creation.

5. *Wholeness Moving Within Itself.* In Unity Consciousness one experiences that all objects in the universe arise from consciousness and are ultimately nothing but consciousness.

Lesson 2
Associations, Modeling Relationships with UML:
Diversifying Self-Referral Relationship to the World of Objects

Wholeness of the Lesson

In the real world, objects have relationships. These manifested relationships appear in many different ways. In this lecture, we explore the range of relationships that can be modeled in UML depending on how the objects' relate to each other.

At the most fundamental level every object is made out of the same essence – and is therefore (in a way) related to everything. An intellectual analysis or model of all these relationships is generally not practical.

A direct experience of the underlying reality of all of manifest creation and our relationship with all of nature is a result of our practice of Transcendental Meditation.

Main Point 1.

Building a software system using OO principles involves an *analysis* step in which the problem is analyzed and broken into pieces as objects are discovered. The pieces are then refined and put together -- in a step of *synthesis* – to give a picture of a unified system. This step of synthesis happens through the identification of relationships between classes, represented by *associations.* This phenomenon is a characteristic of all knowledge – it arises through a combination of analysis and synthesis.

Main Point 2.

Associations model the relationships that can exist between concepts. Simple associations are modeled using an *arrow*.

The arrow can have a name for ease of reading, and additional symbols to indicate direction and multiplicity.

The ends of an association arrow can also specify an association role, which is a different name for the connecting concept that is used in the context of this relationship.

The simplest state of awareness can also be modeled with an arrow from itself to itself.

Main Point 3.

There are several special forms of association, such as reflexive associations, aggregation, composition, and association classes. Although most of these have their own symbols, it would still be possible to model these relationships without them. The use of the symbols is to (easily) communicate additional information about the relationship. Nevertheless, even these additional symbols are still based on the simple concept of an *arrow*. This is an example of diversity on the basis of unity.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

ASSOCIATIONS IN A CLASS DIAGRAM

1. Class diagrams are defined in terms of classes and their relationships (associations)

2. Although there are various special association forms (composition, aggregation, etc.), all are variations of the fundamental concept of an association from one object to another.

---

3. *Transcendental Consciousness* is related to itself through its own self-referral dynamics.

.

4. *Impulses Within The Transcendental Field.* The fundamental relationship that pure consciousness has with itself gives rise to all other forms of relationship.

5. *Wholeness Moving Within Itself.* In Unity Consciousness, one recognizes that the relationship of the Self to the Self is not only fundamental, but is in reality the only relationship there is.

Lesson 3
Inheritance and Composition
Reflecting the Whole in the Part

Wholeness of the Lesson

Inheritance and Composition are types of relationships between classes that support reuse of code. Inheritance makes polymorphism possible, but can lock classes into a structure that is may not be flexible enough in the face of change. Composition is more flexible but does not support polymorphism. Composition and inheritance are techniques based on the principle of preserving sameness in diversity, silence in dynamism.

Main Point 1.

Inheritance is used to model IS-A relationships. Although inheritance offers reuse (the subclass inherits all public and protected methods and attributes), reuse should not be the primary motivation for using inheritance.

The field of pure intelligence is inherited by everyone, and can easily be accessed through the practice of the TM technique.

.

Main Point 2.

Inheritance should only be used when you have a clear IS-A relationship. Otherwise, it is better to use composition because it has better support for change.

Even in clear IS-A relationships, inheritance may not be the best choice because of its inflexibility.

Software relationships that reflect the real world are more natural and easier to understand. Likewise, life in accord with natural law tends to go forward without obstacles; life in violation of natural law tends to be “bumpy”.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

BALANCING REUSABILITY WITH ADAPTABILITY

1. When requirements change, you should implement these changes by adding new code, not by changing old code that already works.

2. Inheritance and Composition are object-oriented principles that support reuse of implementation, but Composition is more flexible.

---

3. *Transcendental Consciousness* is the infinitely adaptable field of pure intelligence that can be 'reused' by every individual at all places, at all times.

4. *Impulses within the Transcendental field.* The "code" that structures manifest life, whose syntax and grammar are impulses of intelligence, has a maximally efficient and compact design, and is reusable and flexible for all time.

5. *Wholeness moving within itself.* In Unity Consciousness, the individual is united with everything else, and inherits the total potential of natural for fulfillment of all desires spontaneously.

Lesson 4
Interaction Diagrams:
Appreciating Dynamism in Silence

Wholeness of the Lesson

In an OO program, objects collaborate with other objects to achieve the objectives of the program. *Sequence diagrams* document the sequence of calls among objects for a particular operation. *Object diagrams* show relationships among objects and the associations between them; they clarify the role of multiple instances of the same class. The principle of *propagation and delegation* clarifies responsibilities of each class and its instances: Requests that arrive at a particular object but cannot properly be handled by the object are *propagated* to other objects; the task is said to be *delegated* to others. Finally, *polymorphism* makes it possible to add new functionality without modifying existing code (as per the *Open-Closed Principle*). In these ways, we use UML diagrams to capture the dynamic features of the system; representing dynamism in the form of a static map illustrates the principle that dynamism has its basis in, and arises within, silence.

Maharishi's Science of Consciousness: In SCI, Maharishi points out that, attempting to know the parts of knowledge separately without the wholeness of knowledge necessarily results in incomplete knowledge. Knowledge of the parts in the context of the whole results in complete knowledge.

Main Point 1.
Sequence Diagrams document the sequence of calls different objects (should) make to accomplish a specific task.

Likewise, harmony exists in diversity: Even though each object is specialized to only perform tasks related to itself, objects harmoniously collaborate to create functionality far beyond each object's individual scope

Main Point 2.
Object Diagrams show the relationships between objects, where each object is an instance of a class, and each reference is represented by a single arrow. This phenomenon illustrates the principle that *the whole is greater than the sum of the parts*: The objects (parts) are not the important focus for an object diagram. What is important is how the objects relate; together, objects and their relationships form a whole that is more than just the sum of individual objects collected together.

Main Point 3.
OO Systems use delegation and propagation. An individual object works only with its own properties, acts only on what it knows, and then asks related objects to do what they know.

When individual actions are on the basis of self-referral dynamics, individual actions are automatically in harmony with each other because all arise from the dynamics of the a single unified field.

Main Point 4.
With polymorphism, objects of a particular type can take many different forms, giving us great power and flexibility.

The Unified Field is the source of all different forms in the universe. The unmanifest can manifest in many different forms.

Main Point 5.
Polymorphism supports use of the *Open-Closed Principle*: the part of our code that is established and tested is closed to modification (change), but at the same time the system remains open to changes, in the form of *extensions.*

In a similar way, progress in life is vitally important, and progress requires continual change and adaptation. But change stops being progressive if it undermines the integrity of life. Adaptability must be on the ground of stability.

CONNECTING THE PARTS OF KNOWLEDGE
WITH THE WHOLENESS OF KNOWLEDGE

OBJECT COLLABORATION IN AN OO SYSTEM

1. Sequence Diagrams and Object Diagrams both show how objects relate to each other.

2. To preserve encapsulation, objects should only act on their own properties, and to accomplish tasks that are the responsibility of other objects, they should send messages (delegation).

---

3. *Transcendental Consciousness* by its very nature, has the fundamental association of self-referral – the Self being aware of the Self.

4. *Impulses within the Transcendental field.* It is by virtue of the self-referral nature of pure consciousness that the structuring impulses of creation emerge to form a blueprint of the universe.

5. *Wholeness moving within itself.* In Unity Consciousness one feels intimately associated with all other things in creation as a result of perceiving all things in terms of one's Self

.

Lesson 5
Inheritance and Abstractions:
Engaging Abstract Levels to Enrich Life

Wholeness of the Lesson

Both abstract classes and interfaces can be used in conjunction with polymorphism, but interfaces provide even more flexibility. Both make possible a variety of implementations or expressions of fundamental themes, the most extreme example of these being the fact that all Java classes inherit from Object. Likewise in the universe, objects form hierarchies of wholeness which express the unmanifest field of pure creative intelligence into all the specific structures of existence and intelligence.

Main Point 1.
When a method is called on a subclass, the JVM by default uses *dynamic binding* to determine the correct method body to execute. Early binding (and hence a slight improvement in performance) can be forced by declaring a method final.

In a similar way, it is said (Maharishi, *Science of Being*) that an enlightened individual need not continually plan and prepare in order to meet the needs of his daily life – instead, the enlightened enjoys spontaneous support of nature, and sees what to do as situations arise. Such individuals are an analogue to "late binding".

Main Point 2.
Abstract classes and Interfaces are both strongly related to the concept of Inheritance. The interface is the most abstract entity in the class diagram, and by programming to interfaces, we generate more flexible code.

Greater abstraction holds the possibility of greater potential; this priniciple is especially evident in the case of the unified field.
.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

ABSTRACTION IN THE FORM OF ABSTRACT CLASSES AND INTERFACES

1. A concrete class embodies a set of concrete behaviors on a set of data whereas an abstract class embodies some concrete behaviors and at the same time gives expression to new, unimplemented behaviors in the form of *abstract methods.*

2. Interfaces (in pre-Java 8) give expression to abstract "unmanifest" behaviors, "pure possibilities," which can be realized in an endless number of ways by implementing classes.

---

3. *Transcendental Consciousness* is a field of all possibilities.

4. *Impulses Within the Transcendental Field.* Pure consciousness, as it prepares to manifest, is a "wide angle lens" making use of every possibility for creative ends.

5. *Wholeness Moving Within Itself.* In Unity Consciousness, awareness is flexible enough to give expression to any possibility that is needed at the time.

Lesson 6
JavaFX and SceneBuilder

Wholeness of the Lesson

JavaFX is a UI library in Java that allows developers to create user interfaces that are rich in content and functionality. The ultimate provider of tools for the creation of beautiful and functional content in manifest existence is pure intelligence itself; all creativity arises from this field's self-interacting dynamics.

Main Point 1.
For creating the look of a JavaFX application, two types of classes are primary: *components* and *containers.* A screen is created by setting the stages with the Stage and Scene container classes. And then the screen is populated with components, like buttons, textboxes, labels, and so on. Components and containers are analgous to the *manifest* and *unmanifest* fields of life; manifest existence, in the form of individual expressions, lives and moves within the unbounded container of pure existence.

Main Point 2.
In JavaFX, components are arranged in a container through the use of *layouts* that organize components in different ways. The most convenient layout is GridPane, which allows you to organize components in a table format, and suffices for most layout needs. For special layout requirements, JavaFX provides half a dozen other layout types. Likewise, all of manifest life is conducted by a vast network of natural laws.

Main Point 3.
A GUI becomes responsive to user interaction (for example, button clicks and mouse clicks) through the event-handling model of JavaFX, in which event sources are associated with EventHandler classes, whose handle method is called (and is passed an Event object) whenever a relevant action occurs. To make use of this event-handling model, the developer defines a handler class, implements the handle method, and, when defining an event source (like a button), registers the handler class with this event source component. The "observer" pattern that is used in JavaFX mirrors the fact that in creation, the influence of every action is felt everywhere; existence is a field of infinite correlation; every behavior is "listented to" throughout creation.

CONNECTING THE PARTS OF KNOWLEDGE
WITH THE WHOLENESS OF KNOWLEDGE

JAVAFX AND SCENEBUILDER

1. In JavaFX, components are placed and arranged in container classes for attractive display.

2. In JavaFX, certain container classes (like HBox) are also considered to be components; this makes it possible to place and arrange container classes inside other container classes. These self-referral dyanmics support a much broader range of possibilities in the design of GUIs.

---

3. *Transcendental Consciousness* is the self-referral field (container) of all possibilities

4. *Impulses Within the Transcendental Field*. The unmanifest foundation of the observable world is the lively self-interaction of impulses of pure consciousness. These are the fundamental components of existence, in orderly arrangement, on the ground of the all-inclusive container – pure consciousness as pure silence.

5. *Wholeness Moving Within Itself*. In Unity Consciousness, one realizes one's own being is the container of all that is.

Lesson 7
Interfaces in Java 8 and the Object Superclass

Wholeness of the Lesson

Java supports inheritance between classes in support of the OO concepts of inherited types and polymorphism. Interfaces support encapsulation, play a role similar to abstract classes, and provide a safe alternative to multiple inheritance. Likewise, relationships of any kind that are grounded on the deeper values at the source of the individuals involved result in fuller creativity of expression with fewer mistakes.

Main Point 1.
Interfaces are used in Java to specify publicly available services in the form of method declarations. A class that implements such an interface must make each of the methods operational. Interfaces may be used polymorphically, in the same way as a superclass in an inheritance hierarchy. Because many interfaces can be implemented by the same class, interfaces provide a safe alternative to multiple inheritance. Java8 now supports static and default methods in an interface, which make interfaces even more flexible: For instance, enums can now "inherit" from other types and new public operations can be added to legacy interfaces without breaking code (as was done with the forEach in the Iterable interface).

The concept of an interface is analogous to the creation itself – the creation may be viewed as an "interface" to the undifferentiated field of pure consciousness; each object and avenue of activity in the creation serves as a reminder and embodiment of the ultimate reality.

Main Point 2.
All classes in Java belong to the inheritance hierarchy headed by the Object class. Likewise, all individual consciousnesses inherit from the single unified field.

Connecting the Parts of Knowledge
With the Wholeness of Knowledge

OBJECT AS THE SUPERCLASS OF ALL

1. Inheritance in Java makes it possible for a subclass to enjoy (and re-use) the features of a superclass

2. All classes in Java – even user defined classes – automatically inherit from the class Object

---

3. *Transcendental Consciousness* is the field of pure awareness, beyond the active thinking level, which is the birthright and essential nature of everyone. Everyone "inherits" from pure consciousness.

4. *Impulses Within the Transcendental Field*. The multiplicity of self-referral impulses that structure the creation are all rooted in, and identical with, pure consciousness itself.

5. *Wholeness Moving Within Itself*. In Unity Consciousness, there is an even deeper realization: The only data and behavior that exist in the universe is that which is "inherited from" pure consciousness – everything in that state is seen as the play of one's own consciousness.

Lesson 8
Functional Programming in Java:
Commanding All the Laws of Nature from the Source

Wholeness of the Lesson

The declarative style of functional programming makes it possible to write methods (and programs) just by declaring *what* is needed, without specifying the details of *how* to achieve the goal. Including support for functional programming in Java makes it possible to write parts of Java programs more concisely, in a more readable way, in a more threadsafe way, in a more parallelizable way, and in a more maintainable way, than ever before.

Just as a king can simply *declare* what he wants – a banquet, a conference, a meeting of all ministers – without having to specify the details about how to organize such events, so likewise can one who is awake to the home of all the laws of nature, the "king" among laws of nature, command those laws and thereby fulfill any intention. The royal road to success in life is to bring awareness to the home of all the laws of nature, through the process of transcending, and live life established in this field.

Main Point 1.
In Java, before Java SE 8, functions were not first-class citizens, which made the functional style difficult to implement. Earlier versions of Java approximated a function with a functional interface; when implemented as an inner class, objects of this type were close approximations to functions. In Java SE 8, these inner class approximations have been replaced by lambda expressions, which capture their essential functional nature – arguments sent to outputs. With lambda expressions, it is now possible to reap many of the benefits of the functional style while maintaining the OO nature of the Java language as a whole.

The "purification" process that made it possible to transform "noisy" one-method inner classes into simple functional expressions (lambdas) is like the purification process that permits a noisy nervous system to have a chance to operate smoothly and at a higher level. This is one of the powerful benefits of the transcending process.

CONNECTING THE PARTS OF KNOWLEDGE
WITH THE WHOLENESS OF KNOWLEDGE

Declarative programming and command of all the laws of nature

1. In Java SE 7, the only first-class citizens are objects, created from classes. The valuable techniques of functional programming and a declarative style can be approximated using functional interfaces.

2. In Java SE 8, functions – in the form of lambda expressions – have become first-class citizens, and can be passed as arguments and occur as return values. In this new version, the advantage of functional programming with its declarative style is now supported in the language.

---

3. *Transcendental Consciousness:* TC, which can be experienced in the stillness of one's awareness through transcending, is where the laws of nature begin to operate – it is the *home of all the laws of nature.*

4. *Impulses Within the Transcendental Field.* As TC becomes more familiar, more and more, intentions and desires reach fulfillment effortlessly, because of the hidden support of the laws of nature

5. *Wholeness Moving Within Itself.* In Unity Consciousness, one finally recognizes the universe in oneself – that all of life is simply the impulse of one's own consciousness. In that state, one effortlessly commands the laws of nature for all good in the universe.

Lesson 9
The Stream API:
Solving Problems by Engaging Deeper Values of Intelligence

Wholeness of the Lesson

The stream API is an abstraction of collections that supports aggregrate operations like `filter` and `map`. These operations make it possible to process collections in a declarative style that supports parallelization, compact and readable code, and processing without side effects.

Deeper laws of nature are ultimately responsible for how things appear in the world. Efforts to modify the world from the surface level only lead to struggle and partial success. Affecting the world by accessing the deep underlying laws that structure everything can produce enormous impact with little effort. The key to accessing and winning support from deeper laws is going beyond the surface of awareness to the depths within.

Main Point 1.

When a Collection is converted to a Stream, it becomes possible to rapidly make transformations and extract information in ways that would be much less efficient, maintainable, and understandable without the use of Streams. In this sense, Streams in Java represent a deeper level of intelligence of the concept of "collection" that has been implemented in the Java language. When intelligence expands, challenges and tasks that seemed difficult and time-consuming before can become effortless and meet with consistent success. This is one of the documented benefits of regular TM practice.

CONNECTING THE PARTS OF KNOWLEDGE
TO THE WHOLENESS OF KNOWLEDGE

LAMBDA LIBRARIES

1. Prior to the release of Java 8, extracting or manipulating data in one or more lists or other Collection classes involved multiple loops and code that is often difficult to understand.

2. With the introduction of lambdas and streams, Java 8 makes it possible to create compact, readable, reusable expressions that accomplish list-processing tasks in a very efficient way. These can be accumulated in a Lambda Library.

---

3. *Transcendental Consciousness* is the field that underlies all thinking and creativity, and, ultimately, all manifest existence.

4. *Impulses Within The Transcendental Field.* The hidden self-referral dynamics within the field of pure intelligence provides the blueprint for emergence of all diversity. This blueprint is formed from compact expressions of intelligence coherently arranged – this blueprint is called the *Veda.*

5. *Wholeness Moving Within Itself.* In Unity Consciousness the fundamental forms out of which manifest existence is structured are seen to be vibratory modes of ones own consciousness.

Lesson 10
Unit-Testing and Exception-Handling:
Progress through Purification of the Path

Wholeness of the Lesson

Test-driven development (TDD) combines traditional coding with unit-testing to ensure that the code that is written is, at the same time, reliable. By testing code as it is written, the need for correcting mistakes later is greatly reduced. When mistakes do arise in Java code, the debugger can be used to track them down. When the code has been written, errors of various kinds may still arise, and to handle these, a robust exception-handling strategy is required. Using the Java SE 8 try-with-resources construct makes it possible to code exception-handling scenarios that were tricky to handle before Java 8.

Main Point 1.

Unit testing, in conjunction with Test-Driven Development, make it possible to steer a mistake-free course as programming code is developed. The self-referral mechanism of anticipating logic errors in unit tests, developed as the main code is developed, is analogous to the mechanism that leads awareness to be established in its self-referral basis; action from such an established awareness is incapable of making a mistake

Main Point 2.

Associated with exception-handling in Java are many well-known best-practices. For example: exceptions that can be caught and handled – *checked exceptions* – reflect the philosophy that, if a mistake can be corrected during execution of an application, this is better result than shutting the application down completely. Secondly, one should never leave a caught exception unhandled (by leaving a catch block empty). Third, one should never ask a catch block to catch exceptions of type Exception because doing so tends to be meaningless.

Likewise, Maharishi points out that, in life, it is better not to make mistakes, but, if a mistake is made, it is best to handle it, to apologize, so that the situation can be repaired; it is never a good idea to simply "ignore" a wrongdoing that one has done. Repairing a wrongdoing requires proper use of speech; an "apology" that does not really address the issue may be too general and may do more harm than good.

CONNECTING THE PARTS OF KNOWLEDGE TO THE WHOLENESS OF KNOWLEDGE

ANNOTATIONS

1. Executing a Java program results in algorithmic, predictable, concrete, testable behavior.

2. Using annotations, it is possible for a Java program to modify itself and interact with itself.

---

3. *Transcendental Consciousness* is the field self-referral pure consciousness. At this level, only one field is present, continuously in the state of knowing itself.

4. *Impulses Within The Transcendental Field.* What appears as manifest existence is the result of fundamental impulses of intelligence within the field of pure consciousness. These impulses are ways that pure consciousness acts on itself, interacts with itself.

5. *Wholeness Moving Within Itself.* In Unity Consciousness, the diversity of creation is appreciated as the play of fundamental impulses of one's own nature, one's own Self.

.

Lesson 11

Java Generics:

Weaving the Universal into the Fabric of the Particular

Wholeness of the Lesson

Java generics facilitate stronger type-checking, making it possible to catch potential casting errors at compile time (rather than at runtime), and in many cases eliminate the need for downcasting. Generics also make it possible to support the most general possible API for methods that can be generalized. We see this in simple methods like max and sort, and also in the new Stream methods like filter and map. Generics involve type variables that can stand for any possible type; in this sense they embody a universal quality. Yet, it is by virtue of this universal quality that we are able to specify particular types (instead of using a raw List, we can use List<T>, which allows us to specify a list of Strings – List<String> -- rather than a list of Objects, as we have to do with the raw List). This shows how the lively presence of the universal sharpens and enhances the particulars of individual expressions. Likewise, contact with the universal level of intelligence sharpens and enhances individual traits.

Main Point 1.

Generic methods make it possible to create general-purpose methods in Java by declaring and using one or more type variables in the method. This allows a user to make use of the method using any data type that is convenient, with full compiler support for type-checking. Likewise, when individual awareness has integrated into its daily functioning the universal value of transcendental consciousness, the awareness is maximally flexible, able to flow in whatever direction is required at the moment, free of rigidity and dominance of boundaries.

Main Point 2.

The Get and Put Rule describes conditions under which a parametrized type should be used only for reading elements (when using a list is of type `? extends T`), other conditions under which the parametrized type should be used only for inserting elements (when using a list of type `? super T`), and still other conditions under which the parametrized type can do both (when no wildcard is used). The Get and Put principle brings to light the fundamental dynamics of existence: there is dynamism (corresponding to Put); there is silence (corresponding to Get) and there is wholeness, which unifies these two opposing natures (corresponding to Both).

CONNECTING THE PARTS OF KNOWLEDGE
TO THE WHOLENESS OF KNOWLEDGE

GENERIC PROGRAMMING USING JAVA'S GENERIC METHODS

1. Using the raw Lists of pre-Java 1.5, one can accomplish the generic programming task of swapping two elements in an arbitrary list using the signature `void swap(List, int pos1, int pos2)`. Using this swap method requires the programmer to recall the component types of the List, and there are no type checks by the compiler.

2. Using generic Lists of Java 1.5 and the technique of wildcard capture, it is possible to swap elements of an arbitrary List with compiler support for type-checking, using the following signature:

```
<T> void swap(List<?> list, int pos1, int pos2)
```

---

3. *Transcendental Consciousness* is the universal value of the field of consciousness present at every point in creation.
4. *Impulses Within the Transcendental Field.* The presence of the transcendental level of consciousness within every point of existence makes individual expressions in the manifest field as rich, unique, and diversified as possible.
5. *Wholeness Moving Within Itself.* In Unity Consciousness, life is appreciated in the fullest possible way because the source of both unity and diversity have become a living reality.