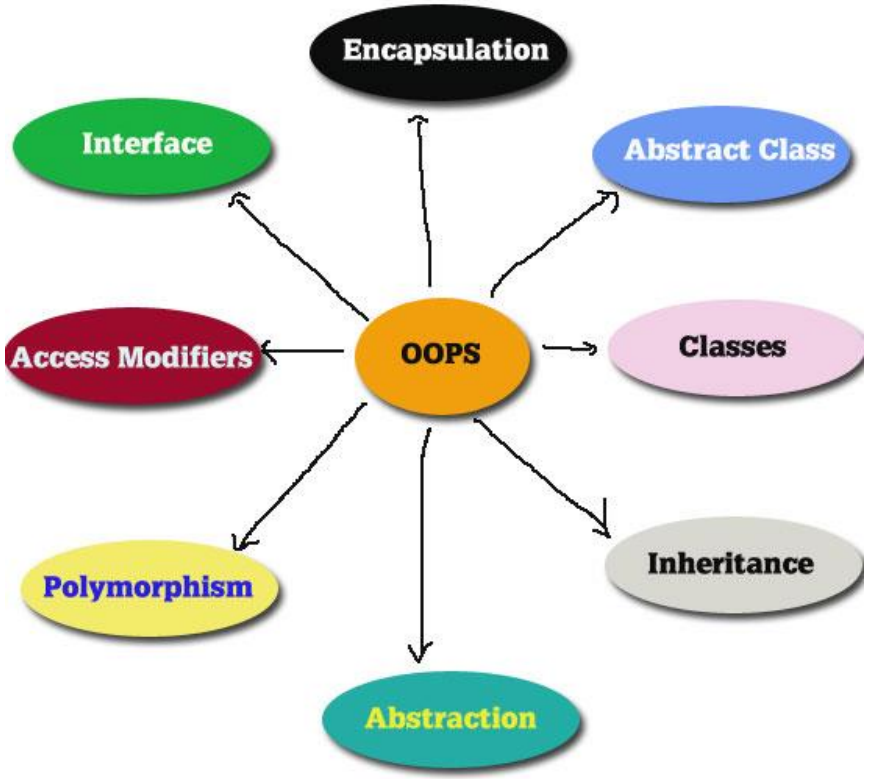
# Java Programming AP Edition
## U4C10 Object-Oriented Thinking

DESIGN OF CLASSES

ERIC Y. CHOU, PH.D. IEEE SENIOR MEMBER

# Object-Oriented Programming
## (scope, visibility modifiers, static modifiers, and final modifier)




**Static Variable/Static Method:**

Constants: (final static variables)

Class Variable: (static variables)

Utility Method: (static methods)

**Instance Variable/Instance Method:**

encapsulated data fields: (private data)

un-protected data: (public data)

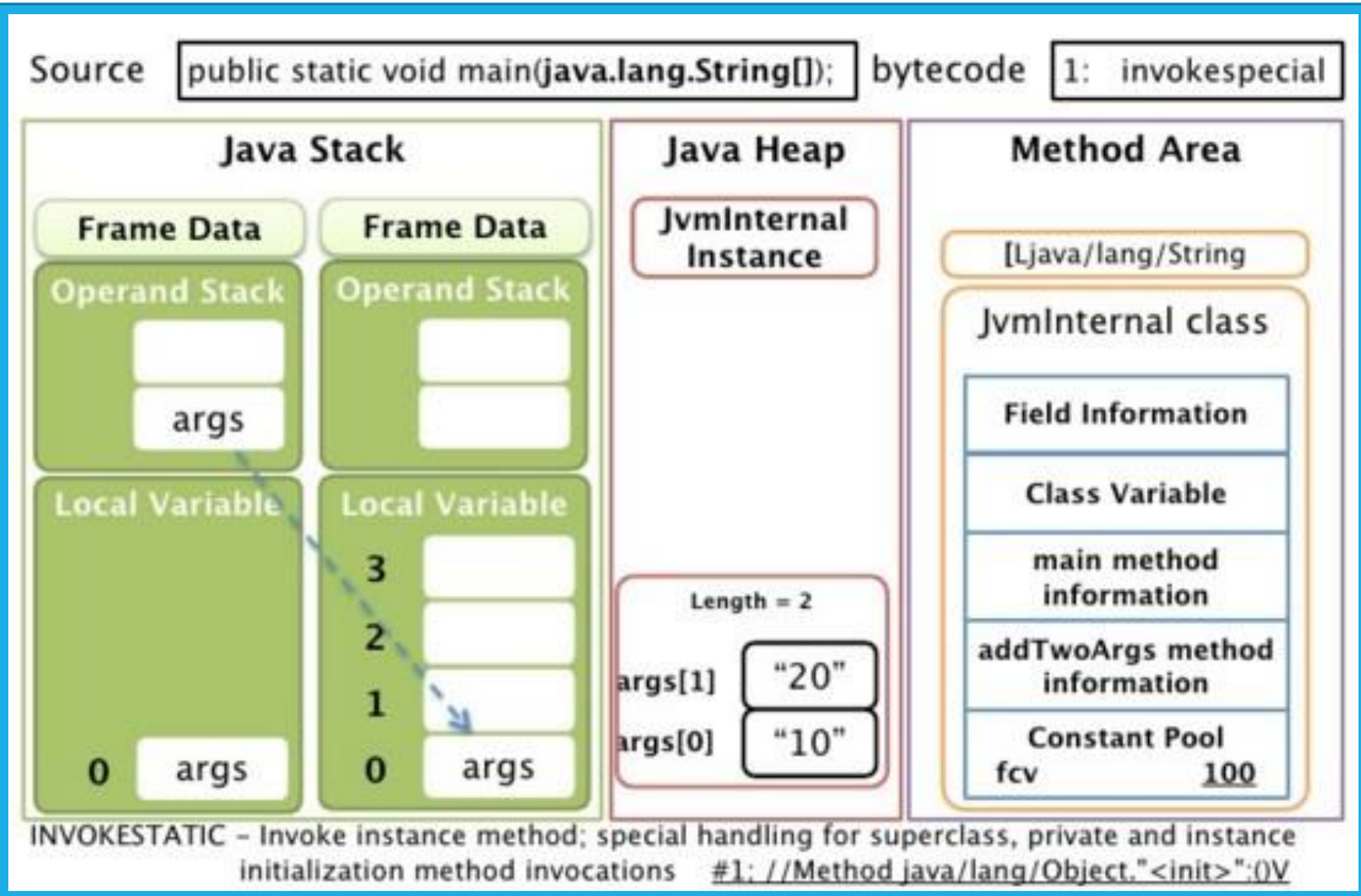
accessor/mutator methods: (public method)

client methods: (private method)

**Encapsulated Data Class:** private data/public method

**Immutable Data Class:** private data/no mutator methods/ no returned pointer

# Memory Allocation



# Object-Oriented Programming

## (scope, visibility modifiers, static modifiers, and final modifier)

**Static Variable/Static Method:**

Constants: (final static variables)

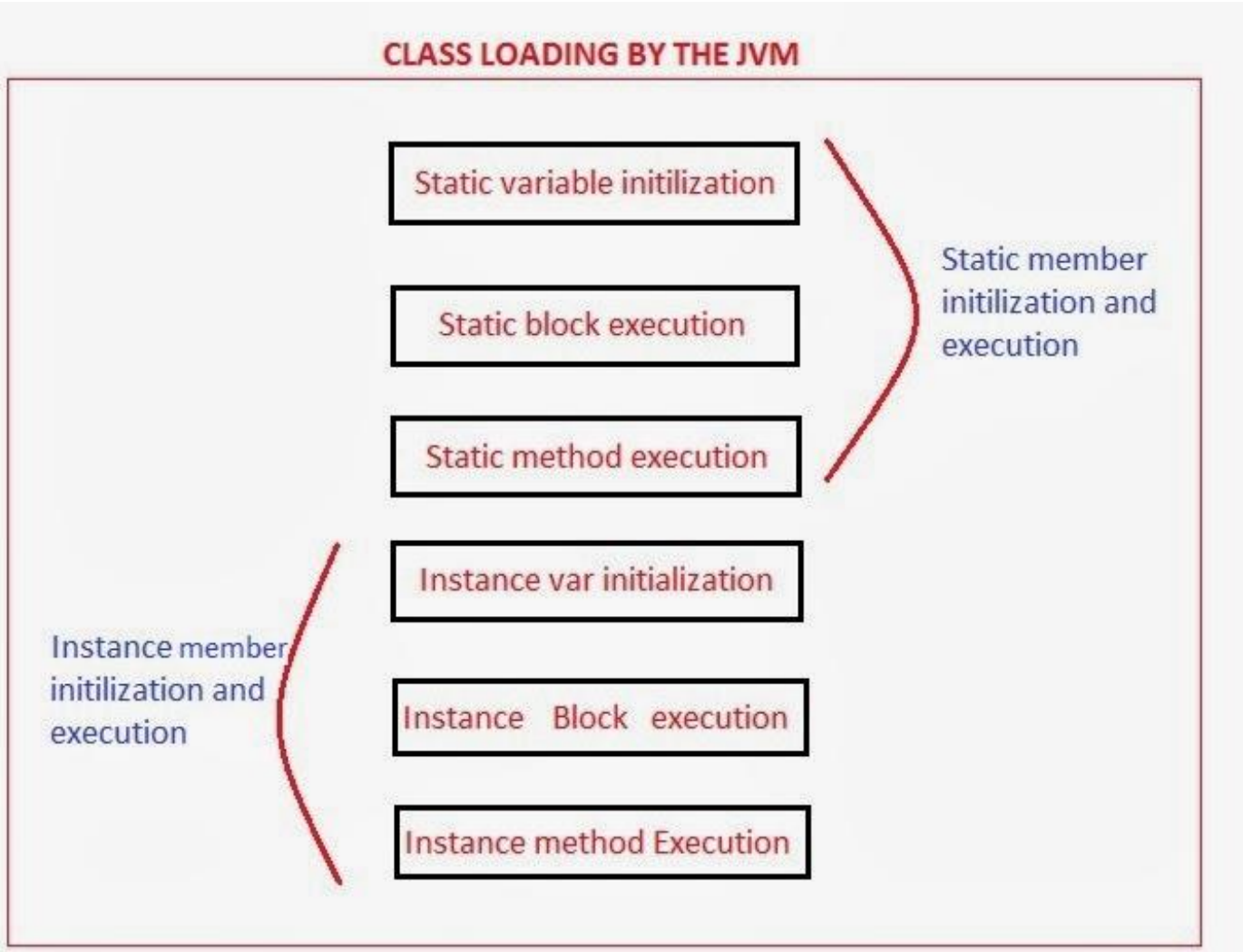
Math.PI, Integer.MIN_VALUE

Class Variable: (static variables)

Circle.count

Utility Method: (static methods)

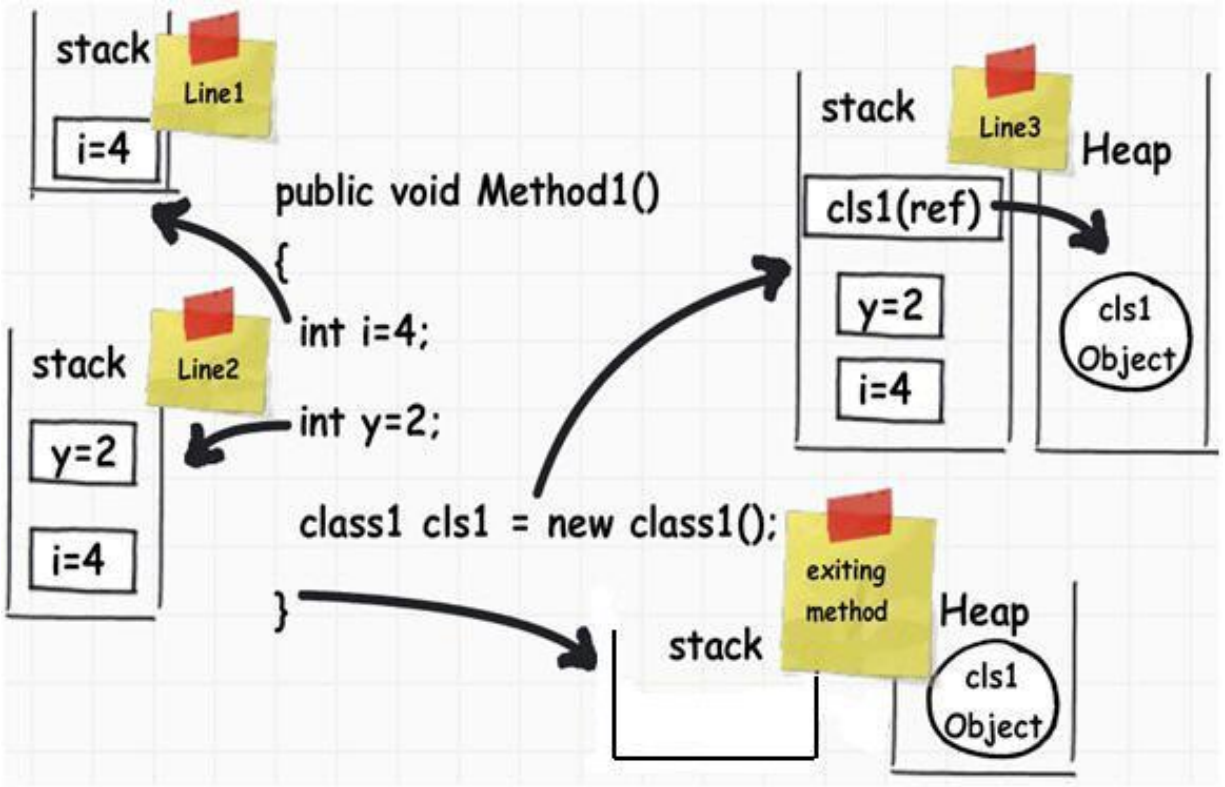
Math.random(), Math.abs(), Integer.parseInt()



# Object-Oriented Programming
## (scope, visibility modifiers, static modifiers, and final modifier)

**Instance Variable/Instance Method:**

encapsulated data fields: (private data)
un-protected data: (public data)
accessor/mutator methods: (public method)
client methods: (private method)



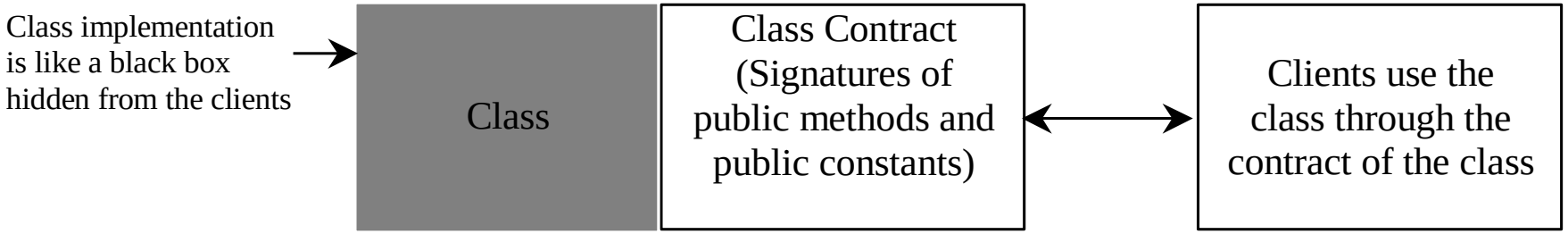

# Object-Oriented Thinking

Part 1-Chapters 1-8 introduced fundamental programming techniques for problem solving using loops, methods, and arrays. The studies of these techniques lay a solid foundation for object-oriented programming. Classes provide more flexibility and modularity for building reusable software.

This chapter reviews chapter 9 and improves the solution for a problem introduced in Part-1 using the object-oriented approach. From the improvements, you will gain the insight on the differences between the procedural programming and object-oriented programming and see the benefits of developing reusable code using objects and classes.


# Class Abstraction and Encapsulation

**Class abstraction** means **to separate class implementation from the use of the class**. The creator of the class provides a description of the class and let the user know how the class can be used. The user of the class does not need to know how the class is implemented. The detail of implementation is encapsulated and hidden from the user.




# Designing a Class

(**Coherence**) A class should describe a **single entity**, and all the class operations should logically fit together to support a coherent purpose. You can use a class for students, for example, but you should not combine students and staff in the same class, because students and staff have different entities.

e.g. Student, Subject, ScoreSheet, Card, Deck, and Hand


# Designing a Class, cont.

(**Separating responsibilities**) A single entity with too many responsibilities can be broken into several classes to separate responsibilities.

The classes String, StringBuilder, and StringBuffer all deal with strings, for example, but have different responsibilities. The String class deals with immutable strings, the StringBuilder class is for creating mutable strings, and the StringBuffer class is similar to StringBuilder except that StringBuffer contains synchronized methods for updating strings.


# Designing a Class, cont.

Classes are designed for reuse. Users can incorporate classes in many different combinations, orders, and environments. Therefore, you should design a class that imposes **no restrictions** on what or when the user can do with it, design the properties to ensure that the user can set properties in any order, with any combination of values, and design methods to function independently of their order of occurrence.


# Designing a Class, cont.

Provide a public no-arg constructor and override the equals method and the toString method defined in the Object class whenever possible.

Overriding standard methods inherited from Object class.


# Designing a Class, cont.

Follow standard Java programming style and naming conventions. Choose informative names for classes, data fields, and methods.

**Always place the data declaration before the constructor, and place constructors before methods.**

**Always provide a constructor and initialize variables to avoid programming errors.**