Deloitte.

Java Basics

USI CBO CR LAUNCHPAD TRAINING PROGRAM

Java Basics

Context, Objectives, Agenda

Context

- *Java is most widely used programming language designed for use in the distributed environment of the internet.*
- *It is most popular programming language for Android smartphone applications.*

Objectives

- To learn
  - What is Java and its notable features
  - Java language Basics, Class & Packages
  - Object oriented programming concepts
  - Other features such as Abstract Classes, Strings, Autoboxing

Agenda

Topic	Content
Java Fundamentals	• Java Features, JDK, JRE, JVM, JVM Architecture, Object Vs Class, Java programming basics
Java Language Basics	• Data types, Variables & their Scope • Operators, Expressions, Blocks and Statements • Control Flow Statements, Keywords
Classes & Packages	• Classes, Methods, Access Modifiers, Constructors, Nested Classes, Abstract Classes, Packages
Object Oriented Programing Concepts	• Polymorphism, Interfaces, Inheritance and Abstract Classes
Other Important Features	• Strings, Wrapper Classes, Autoboxing, Calendar API

Java Fundamentals

Java Fundamentals

Key Objectives

01 **What is Java and its core features?**

02 **JVM, JRE and JDK**
JVM Architecture

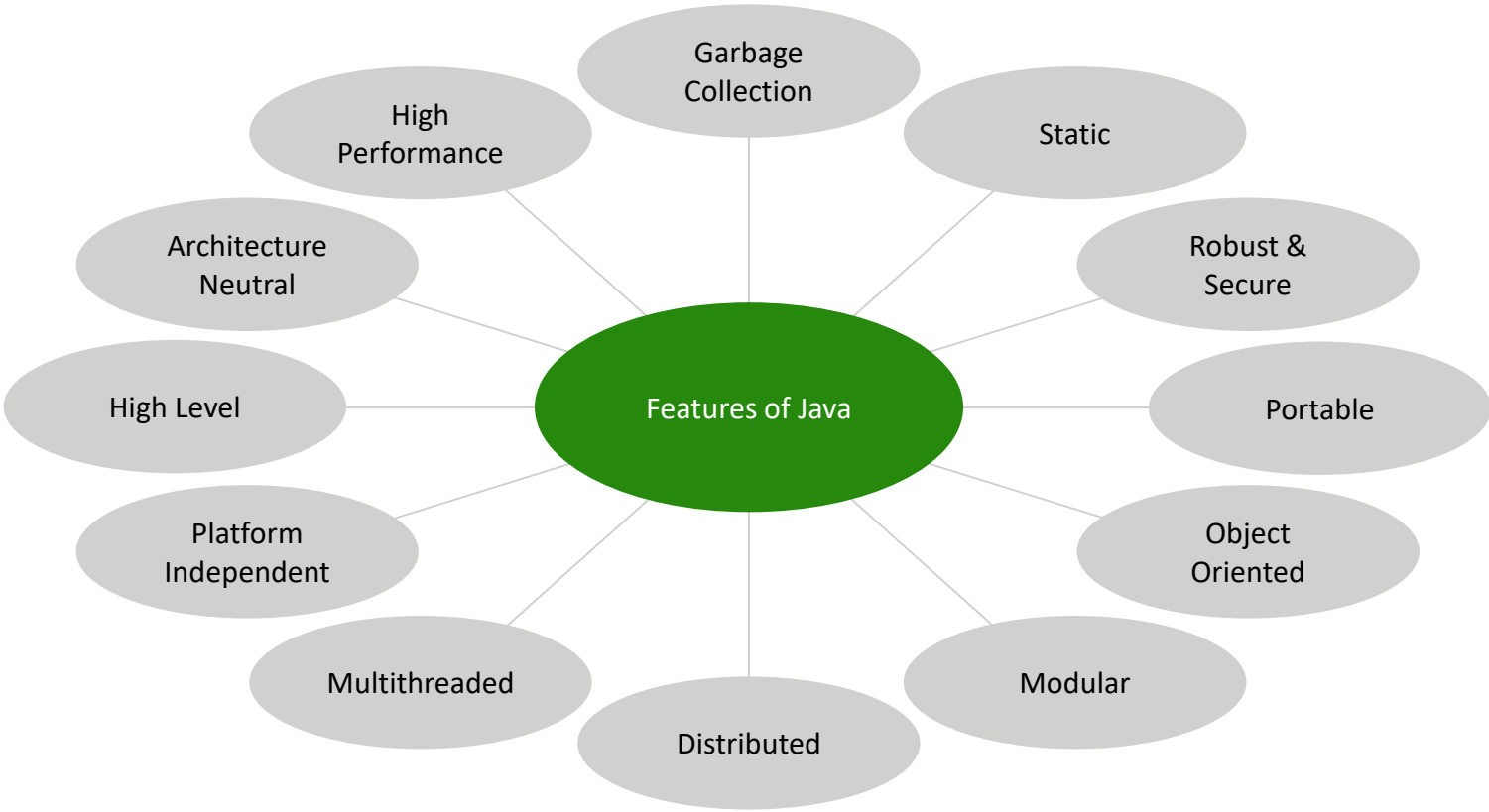
03 **Object vs. Class**

04 **Basic Java program**

Java Fundamentals

Features of Java

Core Features of Java language



Java Fundamentals

Features of Java – at a glance

1. Garbage Collection

- Objects that are no longer used are automatically garbage collected and its memory will be de-allocated

2. Static

- It is used for memory management

3. Robust & Secure

- No explicit pointer
- It uses strong memory management

4. Portable

- Java facilitates you to carry the Java bytecode to any platform. It does not require any implementation

5. Object Oriented

- Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

6. Modular

- It divides into modules (or independent programs)

7. Distributed

- Java makes us able to access files by calling the methods from any machine on the internet.

8. Multi-Threaded

- Java programs deal with many tasks at once by defining multiple threads

9. Platform Independent

- Java code can be run on multiple platforms Ex: Windows, Linux
- The bytecode is platform-independent(Write Once and Run Anywhere(WORA))

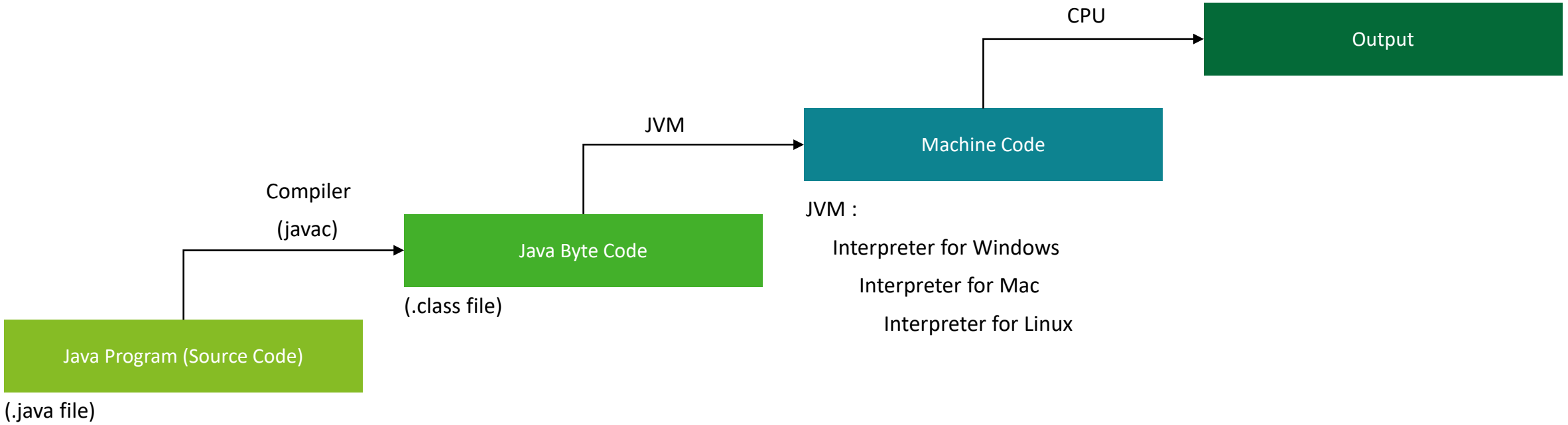
10. High Level

- Java is English like language (If, else, switch)

Java Fundamentals

Working of Java code

How does a Java code work ?



Java Fundamentals

Relationship between JVM, JRE and JDK (Java building blocks)

JVM

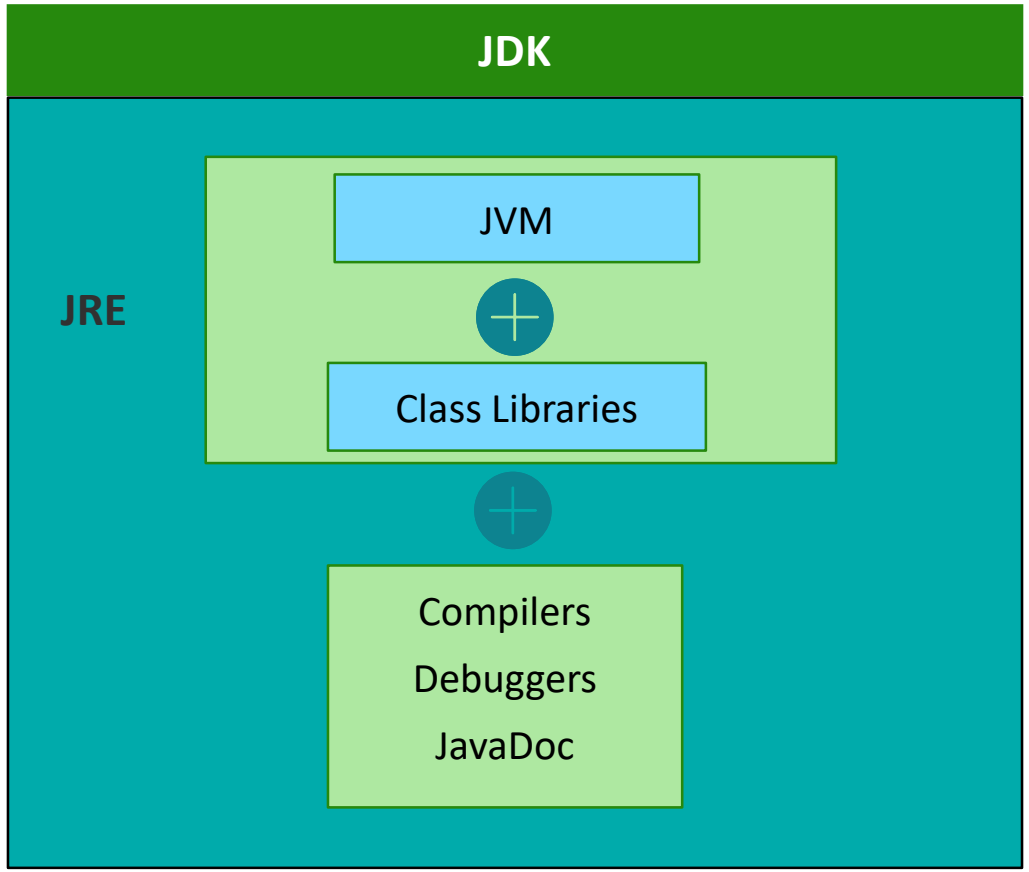
- JVM is a *run time instance* of JRE
- Is an *abstract machine* that enables your computer to run a Java program.

JRE

- Software package that provides *Java class libraries*, *Java Virtual Machine* (JVM), and *other components* required to *run Java applications*
- JRE is a combination of interpreter, binaries and libraries
- Customer uses JRE to run and test a Java application

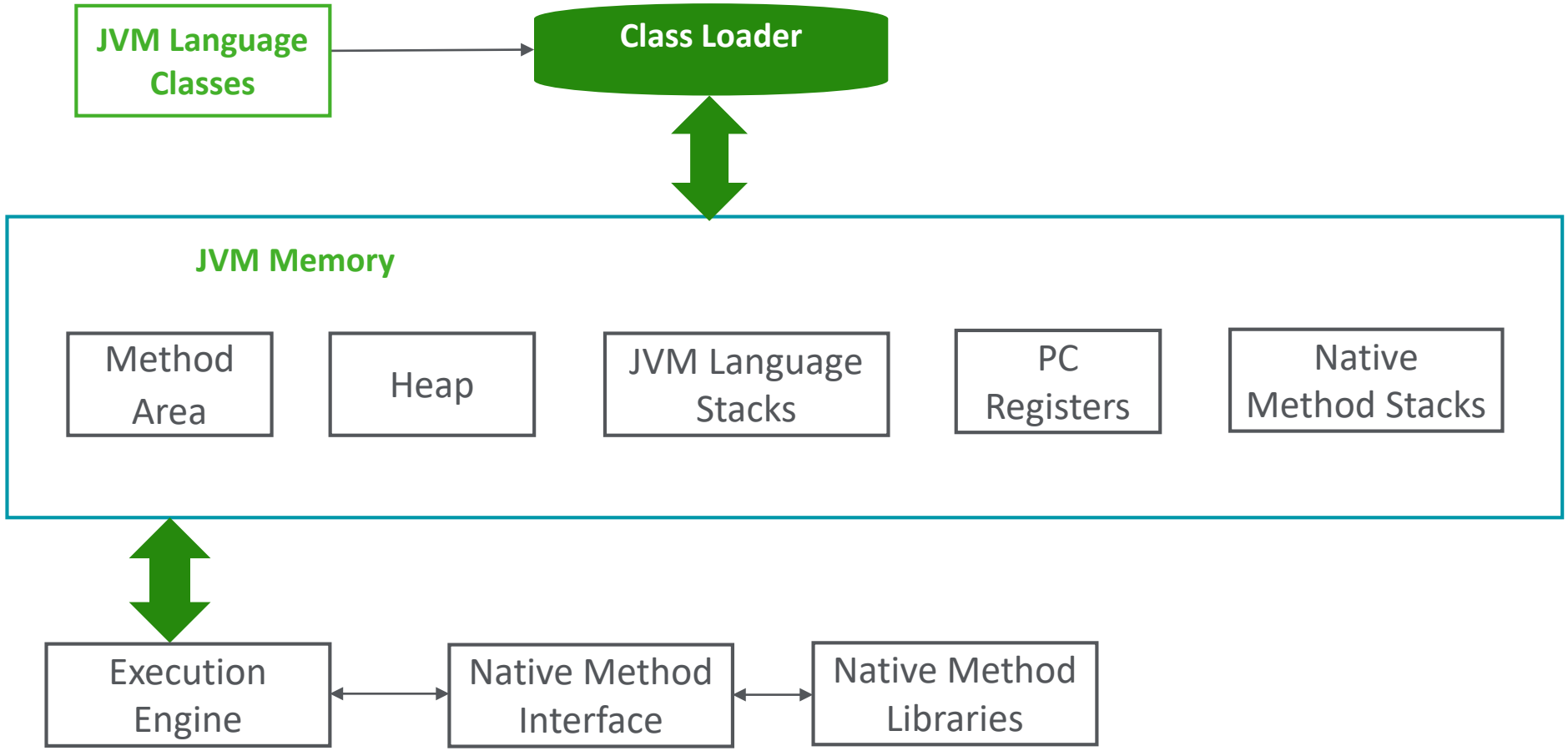
JDK

- Is a software development kit required to develop applications in Java.
- JDK also contains a number of development tools (JRE, Compilers, JavaDoc, Java Debugger, etc).
- Developer uses JDK to develop, compile and build a Java program



Java Fundamentals

JVM Architecture



Java Fundamentals

JVM Architecture - Explained

	JVM Terminology	Description
✓	**Class Loader**	• The class loader is a subsystem of JVM used for loading class files. It performs 3 major functions: *Loading, Linking, and Initialization.* • There are 3 built-in class loaders in Java: *1) Bootstrap 2) Extension 3) System/Application Class loader*
✓	**Method Area**	• JVM Method Area stores class structures like metadata, Class name, Variables, the constant runtime pool, and the code for methods • There is only one method area per JVM, and it is a shared resource.
✓	**Heap**	• All the Objects, their related instance variables, and arrays are stored in the heap. This memory is common and shared across multiple threads. • There is also one Heap Area per JVM. It is also a shared resource.
✓	**JVM language Stacks**	• Java language Stacks store local variables, and it's partial results. Each thread has its own JVM stack, created simultaneously as the thread is created. It plays a part in method invocation and return. • A new frame is created whenever a method is invoked, and it is deleted when method invocation process is complete
✓	**PC Registers**	• PC (Program Counter) register stores the address of the Java virtual machine instruction currently being executed. • Obviously each thread has separate PC Registers.

Java Fundamentals

JVM Architecture – contd..

	JVM Terminology	Description
✓	**Native Method Stacks**	• It contains all the native methods used in the application. • For every thread, separate native stack is created.
✓	**Execution Engine**	• Execution engine execute the .class (bytecode). It reads the byte-code line by line. It contains 3 parts: • *1) Interpreter 2) Just-In-Time Compiler(JIT) 3) Garbage Collector*
✓	**Native Method Interface**	• Java Native Interface (JNI) interacts with the Native Method Libraries written in another language like C, C++ etc. • It enables JVM to call C/C++ libraries and to be called by C/C++ libraries which may be specific to hardware.
✓	**Native Method Libraries**	• It is a collection of the Native Libraries(C, C++) which are required by the Execution Engine.

Java Fundamentals

Java Programming Basics

Key Points to note !!

1. Java program files are saved with .java extension	**6.** Only a corresponding JVM can understand the bytecode
2. The JDK command javac compiles the .java file	**7.** JVM interprets the bytecode to real machine instructions
3. The output from Java compiler is .class file that contains bytecode	**8.** The Real machine executes the instructions to run the program
4. A class may depend on other class for compilation and execution	**9.** Can run on any Operating System (Mac, Windows, Linux)
5. The environment variable class path points to the folders of .class files for the JDK/JVM to refer	**10.** This is how a typical Java program is internally executed

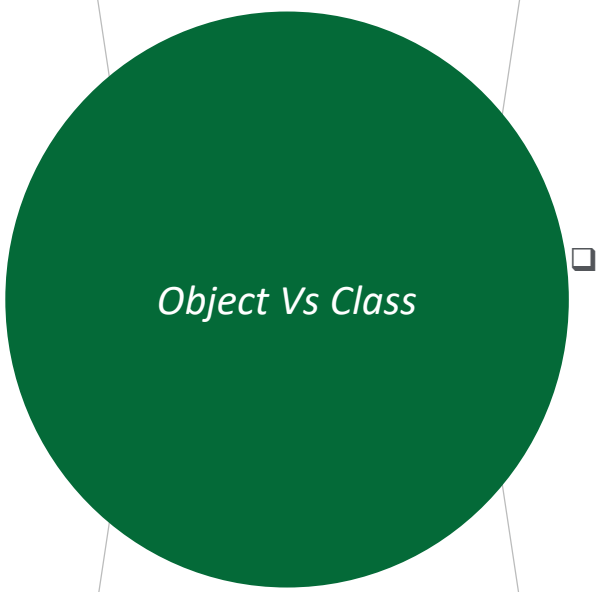
Java Fundamentals

Object vs Class

- An **Object** is a software bundle of related *state and behavior* often used to model the real-world objects
- Objects are key to understanding object-oriented technology
- Software bundle that has a "**state**" and a "**behavior**" (characteristics)
- State – represents data(value) of an object and
- Behavior (functionality) of an object.

- **Ex: Motorbike** – Pulsar (speed, rpm, gear – state | braking, accelerating – behavior)
- An object stores its state in fields (**variables**) and exposes its behavior through methods (**functions**). Thus an object is an encapsulation of state and behavior.

- **Object** is an **instance** of a class
- An object contains an address and takes up some space in memory
- State and behavior of an object are accessed using dot(.) operator



- A **Class** is a collection of objects. It is a logical entity
- A class is the **blueprint** from which individual objects are created
- A **class** can be defined as a **template** that describes the behavior/state that the object of its type support

- **Motorbike example** - The fields cadence, speed, and gear represent the object's state, and the methods (changeCadence, changeGear, speedUp etc.) define its interaction with the outside world

- Class does not consume any space
- Java uses ***class*** keyword to define a class
- Objects of a class are created using ***new*** operator

Java Fundamentals

Basic Java Program Structure

My First Program

```
public class MyFirstJavaProgram {

        public static void main(String[] args) {

        }

}
```

- Every Java program contains at least one class with main() method
- The signature of the main() method is pre-defined
- Names of classes and methods follow camel casing
- Class name starts with Uppercase and method name with Lowercase
- Source file is generally named after the class name

Printing on Console - print() vs. println()

```
public class MyFirstJavaProgram {

        public static void main(String[] args) {

            System.out.println("Hello Java");

        }

}
```

- System.out.*print()* method prints text on the console
- System.out.*println()* method also adds a new line
- These methods take a String as a parameter
- The String must be enclosed in double quotes

Knowledge Check

Q1. What makes Java independent?

- Java Development Kit
- Java Virtual Machine
- Java Runtime Environment
- None of the above

Q2. In Eclipse, which folder contains the actual source code of a Java application?

- bin under project folder
- src under project folder
- doc under project folder
- None of the above

Q3. Which of the following is not a Java features?

- Dynamic
- Architectural Neutral
- Use of pointers
- Object-oriented

Q4. A team wants to set up dev environment and test environment for their application development work using Java. Which of the following needs to be installed as part of the test env?

- Both JDK and JRE
- Only JDK
- Only JRE
- Only Eclipse

Break – 15 min.

Java Language Basics

Java Language Basics

Key Objectives

01 **Data Types, Variables and Scope**

02 **Operators, Expressions, Blocks and Statements**

03 **Control Flow Statements**

04 **Keywords**

Java Language Basics

Data Types in Java

What is a Data Type?

Data types specify the different sizes and values that can be stored in the variable

1. Primitive Data types

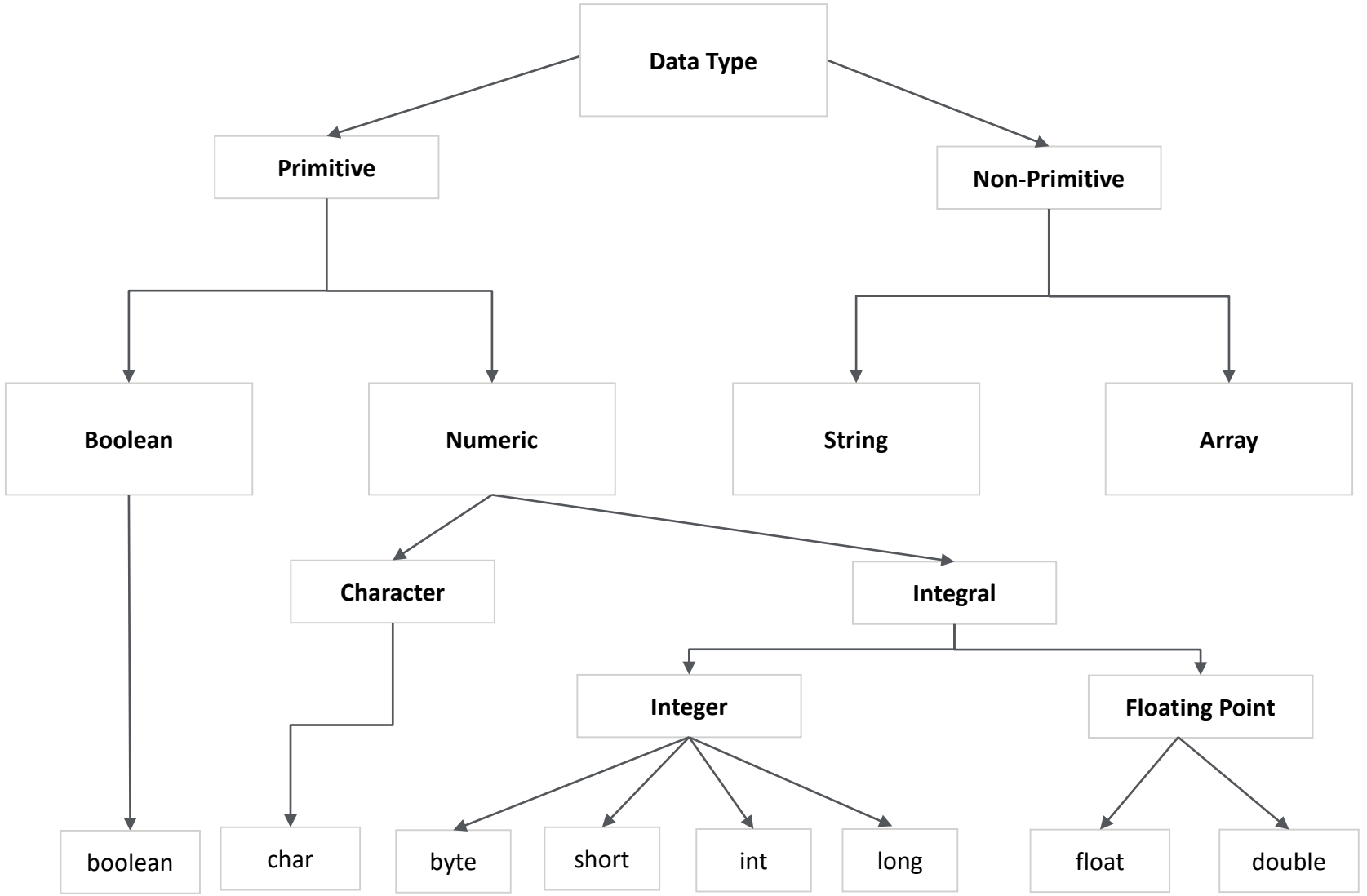
The primitive data types(8) include boolean, char, byte, short, int, long, float and double

2. Non-Primitive Data Types

The non-primitive data types include Classes, Interfaces, and Arrays

Java Language Basics

Hierarchical representation of Data Types



Java Language Basics

Variables and their Scope

Java is statically typed language

Every variable must be declared along with it's type

The type of variable never allowed to be changed

Variables names follow camel case

Variables names starts with lower case letter

Variables can be assigned with values at the time of declaration

Example:

```
// declare variables
 int firstNumber = 10;
 int secondNumber = 20;
 // process data
 int sum = firstNumber + secondNumber;
 // print data
 System.out.println(sum);
```

There are ***3 types of variables*** in Java:

Local variable : declared inside the body of the method

Instance variable (Non-static fields) : declared inside the class but outside the body of the method

Class variable (Static Fields) : a variable that is declared as static

```
class A {

int data = 50; // instance variable
static int m = 100; // static variable

void method() {
int n = 90; // local variable
}
} // end of class
```

Java Language Basics

Operators

Arithmetic Operators

- Addition(+), Subtraction(-), Multiplication(*), Division(/)
- Modulus(%), Post or Pre increment(++),
  Post or Pre decrement(--), Assignment(=)

Comparison Operators

- Equals(=), Not equals(!=), Less than(<), Greater than(>)
- Lesser or equal to(<=), Greater or equal to(>=)

Bitwise Operators

- Bitwise OR(|), Bitwise AND(&), Bitwise XOR(^), Bitwise Compliment(~)
- Signed Right Shift(>>), Signed Left Shift(<<), Unsigned Right Shift(>>>), Unsigned Left Shift(<<<)

Logical and Other Operators

- Logical OR(||), Logical AND(&&)
- Sting Concatenation(+)

Java Language Basics

Expressions and Blocks

❑ Expressions

- A construct made up of variables, operators & method invocations
- The entire construct that evaluates to a single value is an Expression

Example:

```
int numGears = 6;

System.out.println(“No. of gears in the bike are “ + numGears);

int area = length * breadth;

if (value1 == value2)

   System.out.println("value1 == value2");
```

❑ Blocks

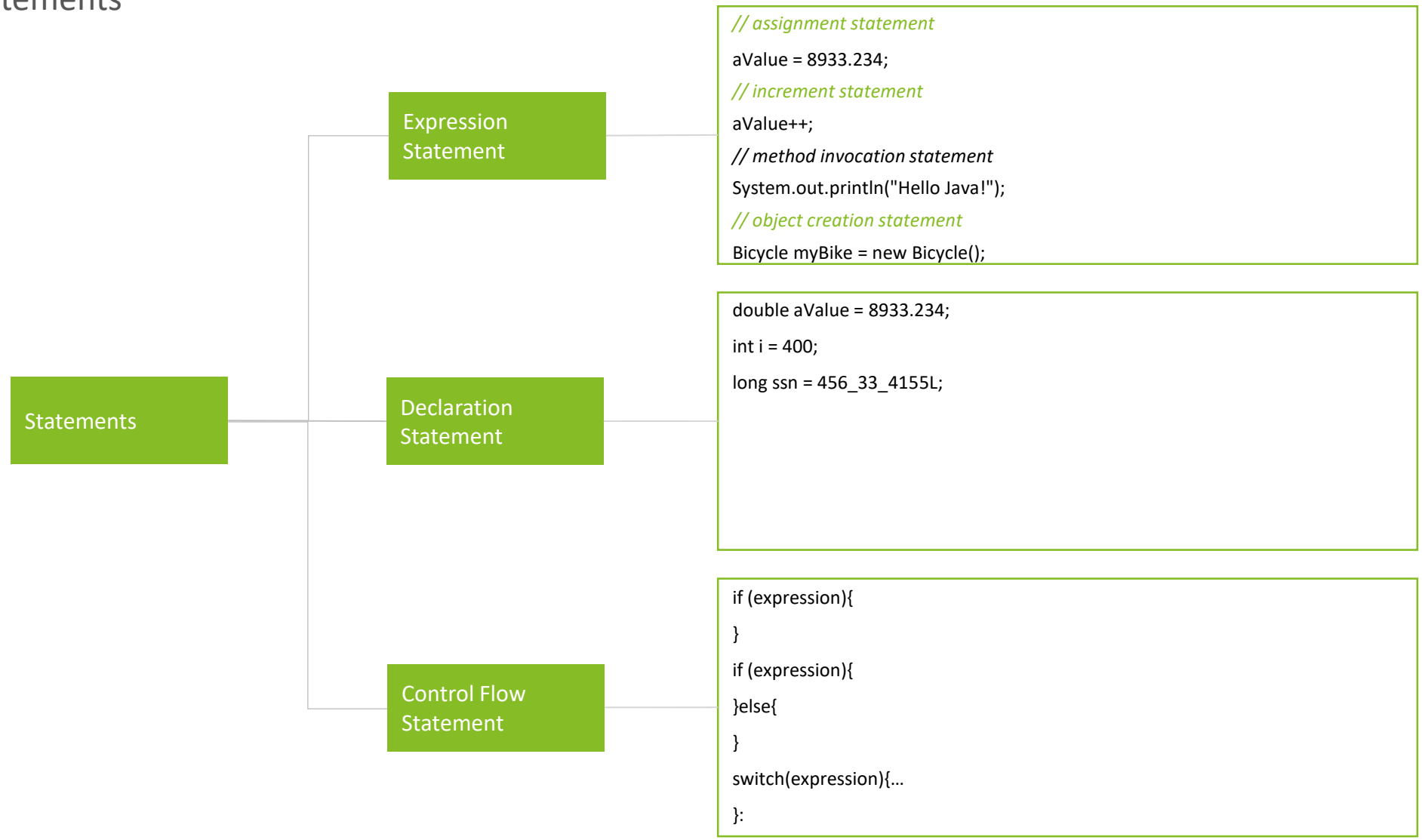
- A Group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed

Example:

```
boolean condition = true;

if (condition) { // Block 1 - Begins

    System.out.println("Condition is true.");

} // Block 1  - Ends

else { // Block 2 - Begins

    System.out.println("Condition is false.");

} // Block 2  - Ends
```

Java Language Basics

Statements



Java Language Basics

Control Flow Statements

1 Decision Making statements

If-then
If-then-else
Switch

2 Looping Statements

While
Do-While
For

3 Branching Statements

Break
Continue
Return

Java Language Basics

Decision Making Statements (if-then, if-then-else, switch)

If-then

```
void applyBrakes() {
        if(isMoving) {
                currentSpeed--;
        }
}
```

If-then-else

```
void applyBrakes() {
        if(isMoving) {
                currentSpeed--;
        } else {
                System.out.println(“The bike has already stopped.”);
        }
}
```

Switch

```
char grade = 'C';

switch(grade) {

  case 'A' :
                System.out.println("Excellent!");
                break;
  case 'B’ :
                System.out.println("Good Job");
                break;
  case 'C' :
                System.out.println("Well done");
                break;
 case 'D' :
                System.out.println("You passed");
                break;
  case 'F' :
                System.out.println("Better try again");
                break;
  default :
                System.out.println("Invalid grade");
}
```

Java Language Basics

Looping Statements (for, while, do-while)

while

```
int count = 1;

while (count < 11) {

   System.out.println(“I am in while loop”);

   count++;

}
```

do-while

```
int count = 1;

do {

   System.out.println(“I am in do-while loop”);

   count++;

} while (count < 11);
```

The for statement

```
for(int i=1; i<11; i++){

    System.out.println("Count is: " + i);

}
```

```
int[] numbers = {1,2,3,4,5,6,7,8,9,10};

for (int item : numbers) {

                System.out.println("Count is: " + item);

}
```

Java Language Basics

Branching Statements (break, continue, return)

The break statement: Used to terminate a for, while, or do-while loops

```
int count = 1;

while (count < 11) {

   System.out.println(“I am in while loop”);

   count++;

}
```

The continue statement: Skips the current iteration of a *for*, while or do-while

```
int count = 1;

do {

   System.out.println(“I am in do-while loop”);

   count++;

} while (count < 11);
```

The return statement: The statement exits from current method and control flow returns to where the method was invoked.

```
return name; // return the value of in the name variable

}
```

```
return; // does not return a value
```

Java Language Basics

Keywords in Java

Type System	• byte, char, short, int, long, float, double, boolean, enum
Program Control	• if, else, for, while, do, goto, return, switch, continue, break, case
Multi-Threading	• synchronized, transient, volatile
Object Oriented Programming	• class, new, public, private, protected, this • extends, super, interface, implements, abstract, instanceof • package, import
Exception Handling	• try, catch, finally, throw, throws
Literals	• true, false, null
Other Keywords	• assert, final, default, static, void, const, native

Knowledge Check

Q1. How many primitive data types are there in Java?

- 6
- 7
- 8
- 9

Q2. Which of these cannot be declared static?

- class
- object
- variable
- method

Q3. Java is a _____ language.

- Strongly typed
- Weakly typed
- Moderate typed
- None of the above

Q4. Which of these keywords is used to prevent content of a variable from being modified?

- final
- last
- constant
- static

Knowledge Check

Q5. All the variables of class should be ideally declared as?

- private
- public
- protected
- default

Q6. How can a protected modifier be accessed?

- Accessible only within the class
- Only within package
- Accessible within package & outside the package but through inheritance only
- Accessible by all

Q7. Can a class be declared with a protected modifier?

- True
- False

Q8. Which of these are selection statements in Java?

- if()
- for()
- continue
- break

Lunch Break – 45 min.

Classes and Packages

Classes and Packages

Key Objectives

01 **Declaring Classes, Variables, Defining Methods**

02 **Access Modifiers, Constructors, Usage of ‘this’**

03 **Nested Classes**

04 **Packages**

Classes and Packages

Declaring Classes

Class declaration

- There can be only one public class per source code file
- The class name, with the initial letter capitalized by convention
- If the class is part of a package, the package statement must be the first line in the source code file, before any import statements that may be present
- If there are import statements, they must go between the package statement (if there is one) and the class declaration
- If there is not a package statement, then the import statement(s) must be the first line(s) in the source code file
- If there are no package or import statements, the class declaration must be the first line in the source code file
- Import and package statements apply to all classes within a source code file
- A class can implement more than one interface

Example

```
class MyClass {

   // field, constructor, and
   // method declarations

}

class MyClass extends MySuperClass implements YourInterface {

   // field, constructor, and
   // method declarations

}
```

Classes and Packages

Declaring Member Variables

❑ Member Variable declaration

- ✓ Must have a modifier - public, private or protected or no modifiers
- ✓ Must have a field type – int, long, byte, String or reference types e.g. String, Array, Custom class.
- ✓ Must have a field name

Example:

```
private int height = 30;
protected String name = “John Doe”;
public float f1 = 2.44f;
private Bike b1 = new Bike();
```

Classes and Packages

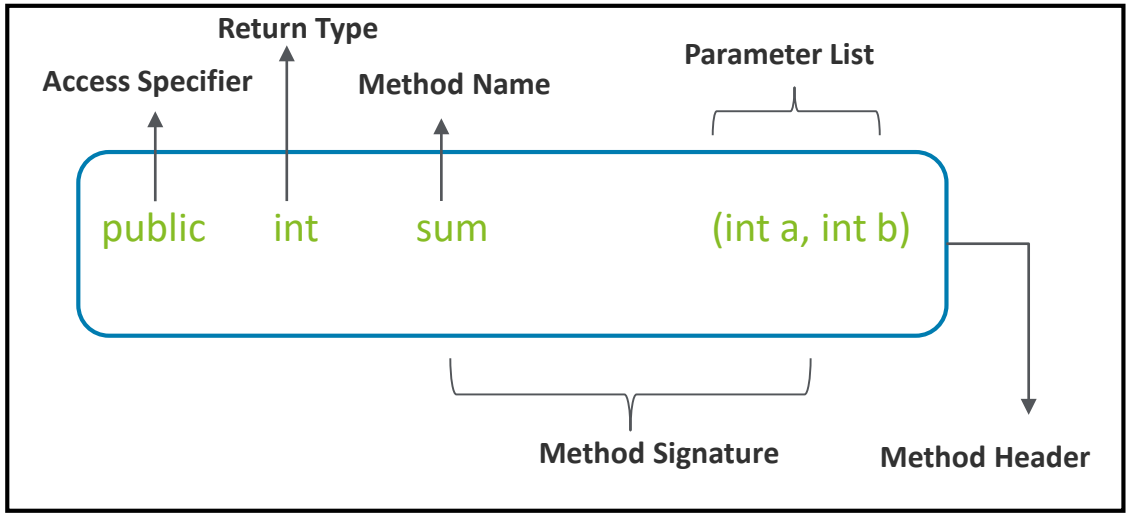
Defining Methods

❑ Method declaration

1. The behavior of a class is defined as a set of methods
2. Must have a modifier - ***public, private or protected*** or no modifiers
3. Must have a **return type** – the data type of the value returned by the method, or void if the method does not return a value.
4. Must have a method name, follows camel case with the first letter in *lower case*
5. Must have a parameter list - a comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses, (). If there are no parameters, you must use empty parentheses.
6. An exception list
7. Must have a method body, enclosed between braces
8. If the methods are accessible outside of the class, they are declared as public
9. Avoid private methods as much as possible

Example:

Method declaration



Classes and Packages

Access Modifiers and Rules

❑ Access Modifiers and Rules

- ✓ 3 access modifiers – public, protected, and private
- ✓ 4 access levels – public, protected, default, and private
- ✓ Classes can have only public or default access

- ✓ Class visibility revolves around whether code in one class can:
  - Create an instance of another class
  - Extend (or subclass) another class
  - Access methods and variables of another class

- ✓ A **default** member may be accessed only if the class accessing the member belongs to the same package
- ✓ A **protected** member can be accessed (through inheritance) by a subclass even if the subclass is in a different package.
- ✓ For a subclass outside the package, the protected member can be accessed only through **inheritance**.
- ✓ Best practices:
  - Make classes public
  - Make fields(state) private
  - Make methods public

Illustration:

Modifier	Class	Package	Subclass
public	Y	Y	Y
private	Y	N	N
no modifier	Y	Y	N
protected	Y	Y	Y

Classes and Packages

Constructors – Things to know!!

- Class name is name of the Constructor as well
- Constructors can accept zero to any number of parameters. A Constructor that do not take any parameters is called *Default constructor*
- Constructors do not return anything; they don't even have a return-type
- Constructors can be Overloaded
- Constructors are never inherited, they cannot be Overridden
- Constructors are special functions to initialize the state at the time of *object creation* itself
- Anywhere in the class, *this* references the current object
- The constructor calls its superclass constructor, which calls its superclass constructor, and so on all the way up to the Object constructor
- Interfaces do not have constructors
- Each superclass in an object's inheritance tree will have a constructor called

Classes and Packages

Usage of ‘this’ keyword

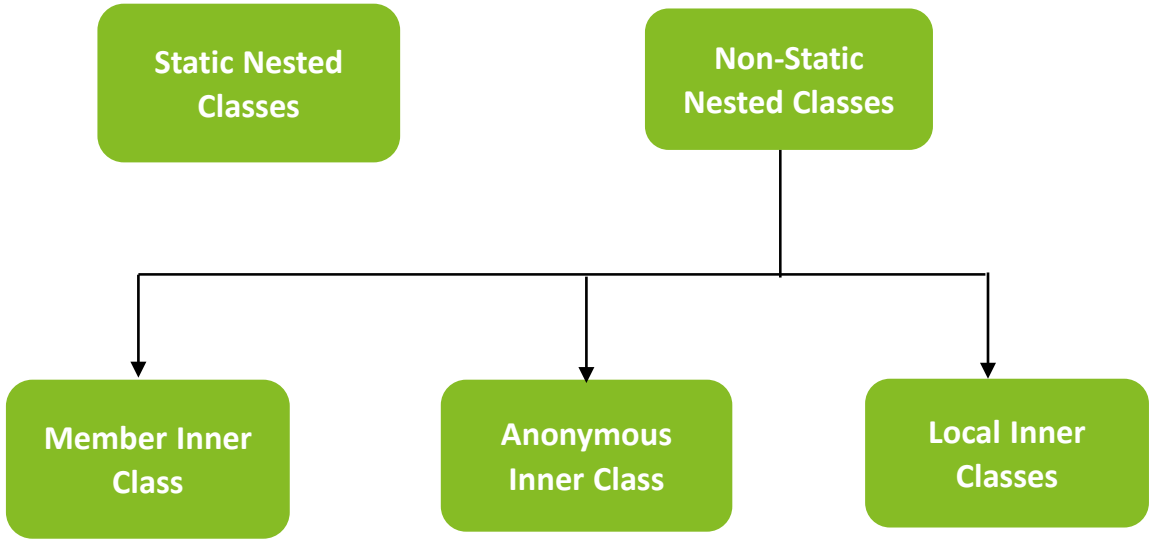
Classes and Packages

Nested Classes

What are Nested Classes?

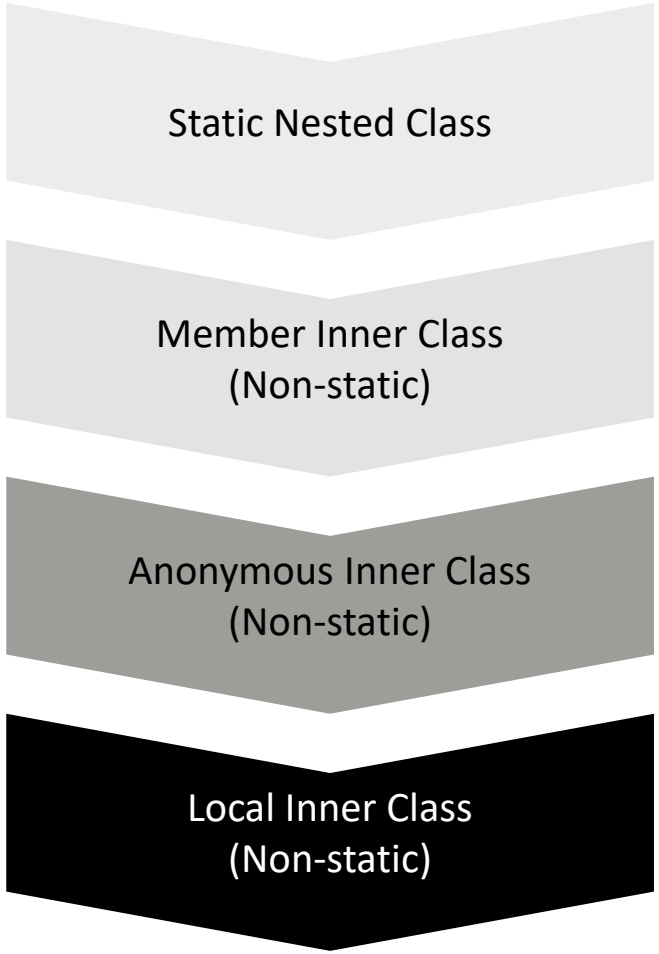
- A class within another class
- Logically grouping of classes that are only used in one place
- Increases encapsulation
- Leads to more readable and maintainable code
- Non-Static Nested Classes a.k.a inner classes

Illustration:



Classes and Packages

Nested Classes



- A static class i.e. created inside a class is called static nested class in java. It can be accessed by outer class name.
  - It can access static data members of outer class including private
  - It cannot access non-static (instance) data member or methods

- A non-static class that is created inside a class but outside a method is called member inner class

- A non-static class that does not have any name
  - Enable you to declare and instantiate a class at the same time
  - They are like local classes except that they do not have a name

- A non-static class defined in a block, which is a group of zero or more statements between balanced braces
  - You typically find local classes defined in the body of a method

Classes and Packages

Abstract classes - Overview

A class that cannot be instantiated is an *Abstract* class
An abstract class can have static methods, main method and constructor

'*abstract*' keyword is used to declare an abstract class
Does not support multiple inheritance

A class with abstract methods must be declared *abstract*

Abstract classes have partial implementation, but meant for further extension by sub classes

Can contain both abstract and non-abstract(concrete) methods
Abstract methods does Not have body where as Concrete methods have body

Can have final, non-final, static and non-static Variables

Abstract and Final Classes – Do you know??

Classes can be final, abstract, or strictfp !

A class cannot be *both final and abstract* !

A *final* class cannot be sub-classed !

An *abstract* class cannot be instantiated !

A single abstract method in a class means the *whole class* must be abstract !

The first concrete class to extend an abstract class must implement *all the abstract methods* !

Classes and Packages

Packages

Package Declaration

- Package declaration must be the *first* statement in a Java class

What is a Package?

- A Package is a collection of classes providing access protection and namespace management
- Classes declare the package to which they belong to

Naming convention

- Package name follows dot notation with an inverted domain name
- E.g.: com.deloitte.myproject, com.mywebsite.newproject

Imports Example

- All the classes from a package can be imported using a wildcard
- E.g. import com.deloitte.myProject.MyCustomClass;,
  import com.deloitte.myProject.*;

Import statement

- Using a class from a different package requires import statement
- *import* statement should always be after package statement

Why Packages?

- Make life easier to find classes, interfaces, etc.
- Avoid naming conflicts
- E.g. java.lang, java.io, java.util, etc.

Knowledge Check

Q1. What is the return type of Constructors?

- int
- float
- void
- None of the above

Q2. Which of the following is a Garbage collection technique?

- Cleanup model
- Mark and sweep model
- Space Management model
- Sweep model

Q3. Which method can be defined only once in a program?

- main method
- finalize method
- static method
- private method

Q4. Which of these can be overloaded?

- Methods
- Constructors
- Both
- None of the above

Knowledge Check

Q5. What is the process by which we can control what parts of a program can access the members of a class?

- Polymorphism
- Abstraction
- Encapsulation
- Recursion

Q6. Which of these methods must be made static?

- main()
- delete()
- run()
- finalize()

Q7. Which of these keywords is used to refer to member of base class from a sub class?

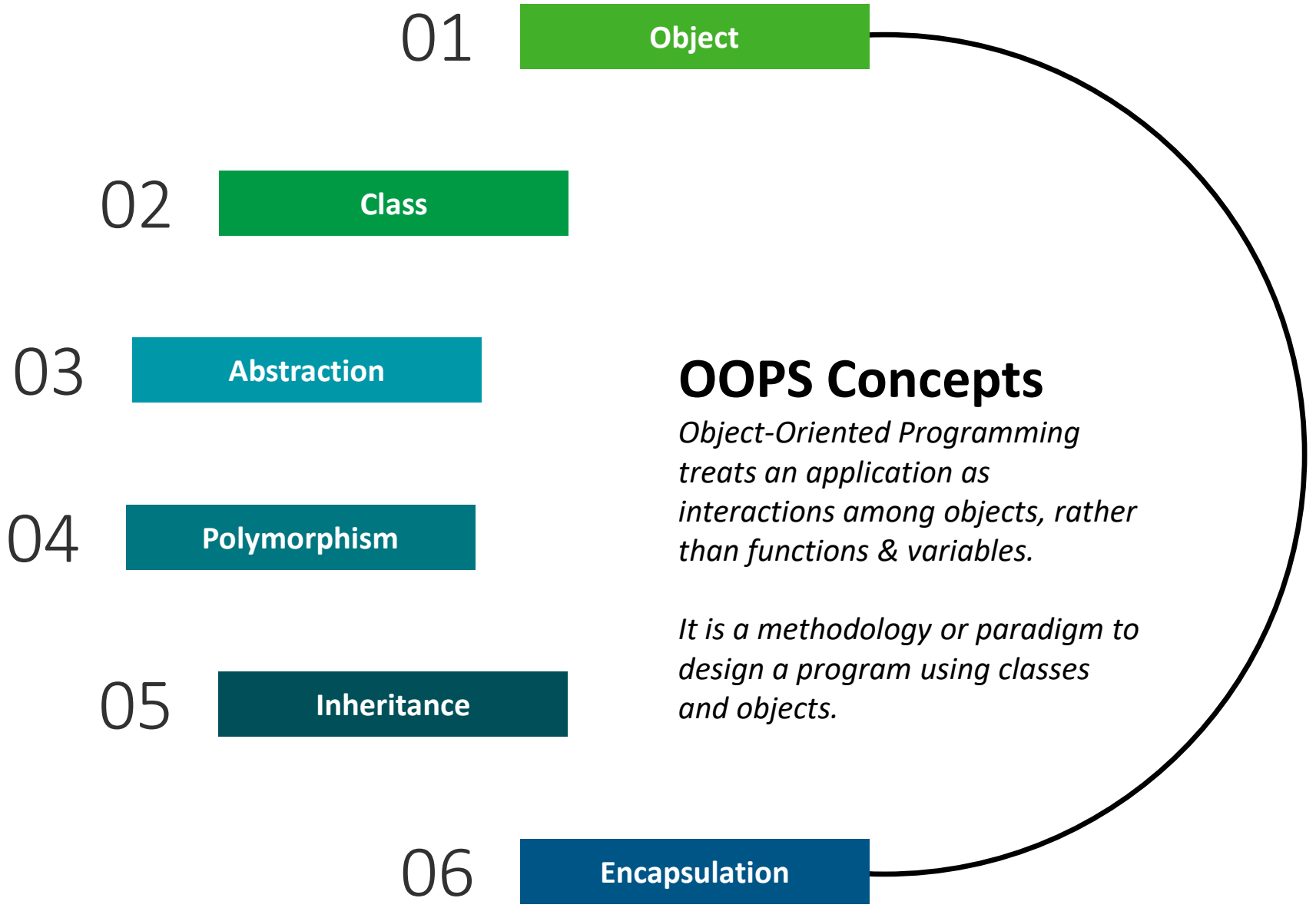
- upper
- super
- this
- None of the above

Q8. What is true about Class.getInstance()?

- it calls the constructor
- it is same as new operator
- it needs to have matching constructor
- it creates object if class does not have any constructor

Object Oriented Programming Concepts

Object Oriented Programming Structures (OOPS)



Object Oriented Programming Concepts

OOPS Principles

Abstraction

Definition

- **Hiding** internal details of an object from the user and showing only relevant data
  - Eg: Logging into an online bank account

How do we achieve it in Java?

✓ Use **abstract class** and interface to achieve abstraction

- Abstract means incomplete (The log in verification process is abstracted from you)

Polymorphism

- The ability of a method to be used in **different** ways, that is, it can take different forms at different times (poly + morphos)
  - There are two types of polymorphism: compile time polymorphism and run time polymorphism

✓ We use method **overloading** and method **overriding**

- Compile time (static) polymorphism occurs when a method is overloaded where as Run time(dynamic) occurs when the actual method itself is changed

Inheritance

- When one object acquires(or **inherits**) all the properties and behaviors of a parent object, it is known as inheritance
  - Inheritance allows us to reuse of code, it improves reusability
    - To inherit a class we use ***extends*** keyword

✓ It is used to achieve **runtime polymorphism**

- The parent class is called the ***base*** class or ***super*** class
- The child class that extends the base class is called the derived class or ***sub*** class or ***child*** class

Encapsulation

- **Binding** object state(fields) and behavior(methods) together
- It protects the integrity of the data
  - Encapsulation is often confused with data abstraction

✓ If you are creating class, you are doing **encapsulation**

- Java bean is the fully encapsulated class because all the data members are private here

Object Oriented Programming Concepts

Method Overloading and Method Overriding - Illustration

Method Overloading:

```
public class Circle {
        public void draw(){
            System.out.println(“Circle with default color");        }
        public void draw(int diameter){
           System.out.println("Circle with default color and diameter");
        }
        public void draw(int diameter, String color){
          System.out.println("Circle with color and diameter");
        }
}
```

Method Overriding:

```
public interface Shape {
        public void draw();
}

public class Circle implements Shape{
        @Override
        public void draw(){
        System.out.println("Drawing circle");
        }

public class Square implements Shape {
        @Override
        public void draw() {
        System.out.println("Drawing Square");
        }
}
Shape sh = new Circle();
sh.draw();

Shape sh1 = getShape(); //logic to determine shape
sh1.draw();
}
```

Object Oriented Programming Concepts

Inheritance - Illustration

Inheritance:

```
class SuperClassA {
        public void foo(){
        System.out.println("SuperClassA");
        } }
class SubClassB extends SuperClassA{
        public void bar(){
        System.out.println("SubClassB");
        }}
public class Test {
        public static void main(String args[]){
        SubClassB a = new SubClassB();
                a.foo();
        a.bar();
        }}
```

Key points:

- ✓ We use extends keyword in java to implement inheritance
- ✓ The object that is getting inherited is called the superclass and the object that inherits the superclass is called a subclass
- ✓ A class can extend only one super class
- ✓ Java does not support multiple inheritance, however a class can implement multiple interfaces

Object Oriented Programming Concepts

Abstract Class - Illustration

Abstract Class:

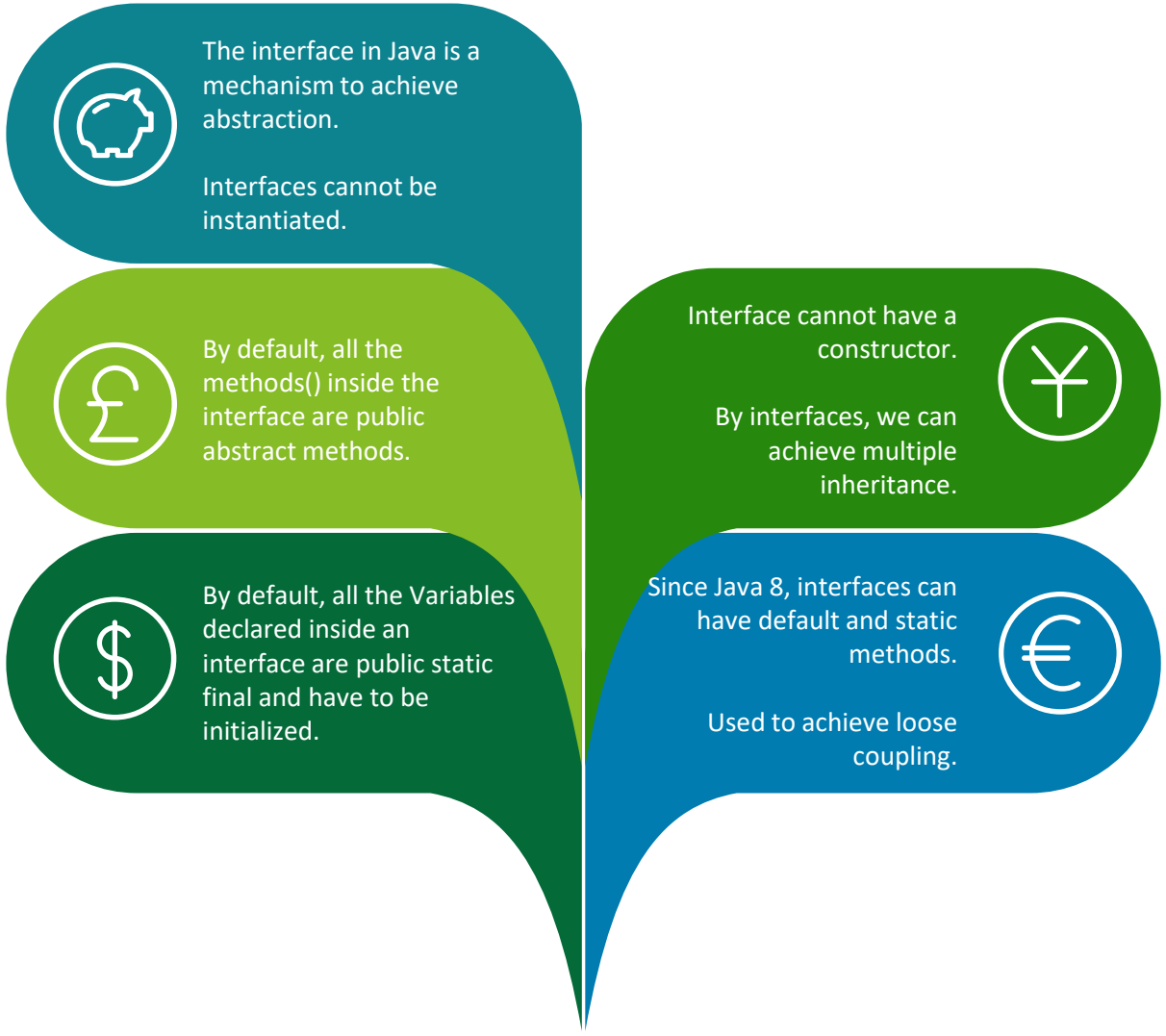
```
interface Shape {
    public double getArea();
    public double getCircumference();
    public void print();
}
abstract class AbstractShape implements Shape {
     public void print() {
          System.out.println("Area: " + getArea());
          System.out.println("Circumference: " + getCircumference());
      }
}
```

Contd..

```
class Rectangle extends AbstractShape {
    private double length;
    private double breadth;

    public Rectangle(double length, double breadth) {
        this.length = length;
        this.breadth = breadth;
    }

    public double getArea() {
        return length * breadth;
    }

    public double getCircumference() {
        return 4 * length;
    }

}
```

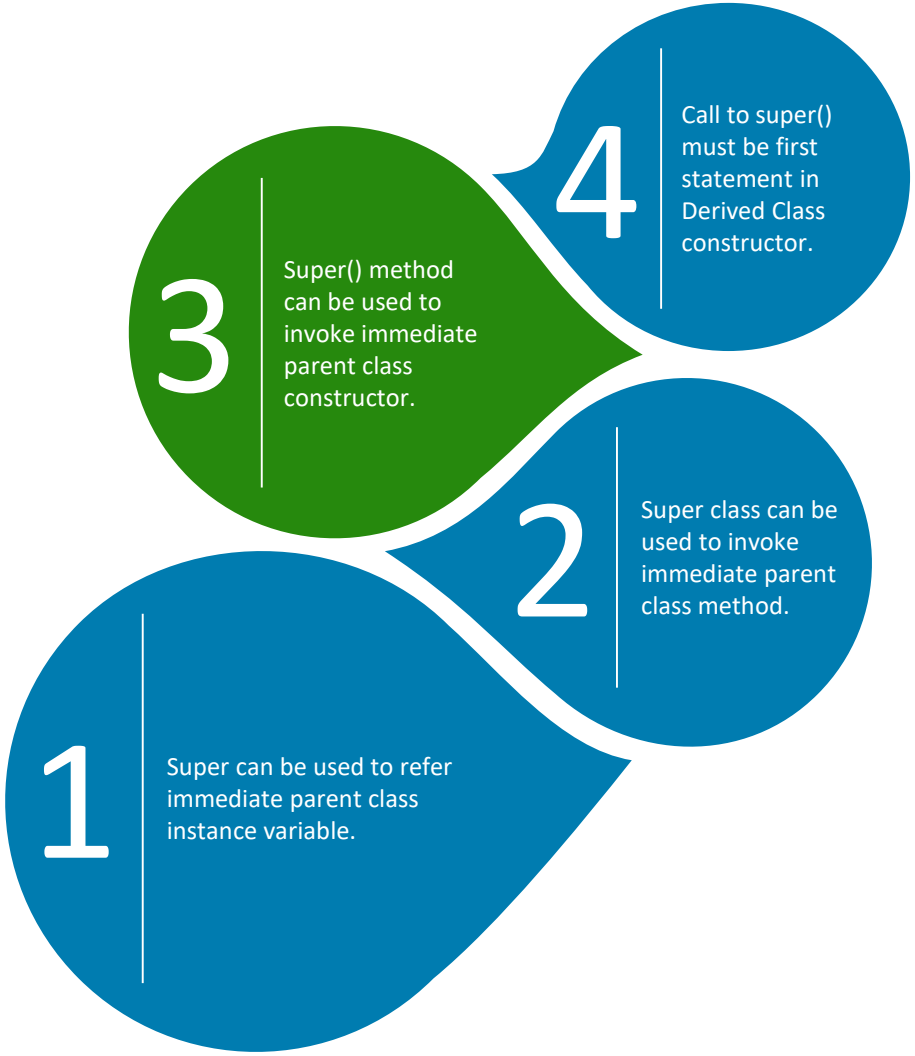
Object Oriented Programming Concepts

Interfaces



Object Oriented Programming Concepts

Usage of Super keyword



Break – 15 min.

Other Important Features

Other Important Features

Key Objectives

01 **Strings and its Operations**

02 **Wrapper Classes**

03 **Autoboxing**

04 **Calendar API**

Other Important Features

Strings and its Operations

Strings

- ✓ Java strings are not primitive types
- ✓ Java strings are objects and are immutable
- ✓ ***There are 2 ways to create a String in Java***:

  a) Using String literal E.g.: String name = “Java Language”;

  b) Using new keyword E.g.: String name = new String(“Java Language”);

Example:

Converting the case	• name.toLowerCase() • name.toUpperCase()
Concatenating strings	• String str = name + “ is platform independent”;
Replacing parts of string	• name.replace(“Java”, “Java Programming”);
Converting a string to array	• name.getBytes() • name.getCharsArray()
Converting primitives to strings	• String.valueOf(123) • String.valueOf(true)
Parsing a string for primitive type	• Integer.parseInt(“123”)

Other Important Features

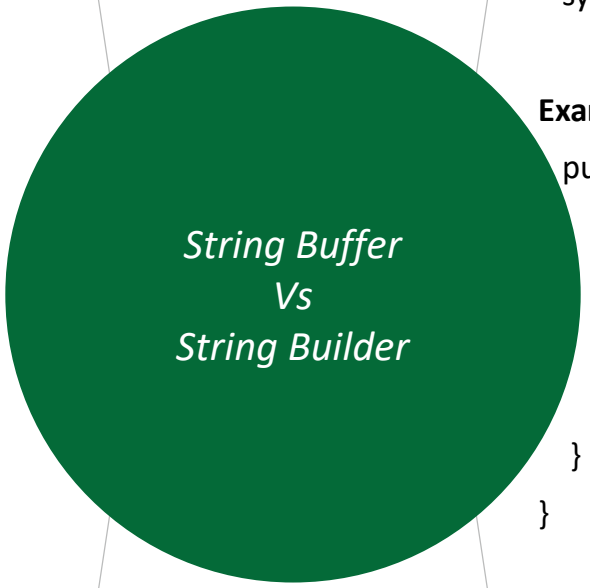
String Buffer Vs. String Builder

String Buffer

- StringBuffer operations are thread safe and synchronized
- StringBuffer performance is lower

Example:

```
public class BufferTest{
   public static void main(String[] args){
      StringBuffer buffer=new StringBuffer("hello");
      buffer.append(“world");
      System.out.println(buffer);
   }
}
```



String Builder

- StringBuilder operations are not thread safe
- StringBuilder performance is faster because there is no synchronization here

Example:

```
public class BuilderTest{
   public static void main(String[] args){
      StringBuilder builder=new StringBuilder("hello");
      builder.append(“world");
    System.out.println(builder);
   }
}
```

Other Important Features

Wrapper Classes / Autoboxing & Unboxing

Wrapper classes

- ✓ Convert primitive into object and object into primitive
- ✓ Since Java 5, autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically
- ✓ The automatic conversion of primitive into an object is known as autoboxing and vice-versa is unboxing

Autoboxing & Unboxing

- ✓ Constructing the Object using new operator is Autoboxing
- ✓ Extracting the Value from Object is Unboxing

Example:

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
Int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Other Important Features

Calendar API

Calendar Class

- Calendar class in Java is an ***abstract class*** that provides methods for converting date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc
- It inherits Object class and implements the Comparable, Serializable, Cloneable interfaces
- Since it is an Abstract class, so we cannot use a constructor to create an instance
- Instead, we will have to use the static method ***Calendar.getInstance()*** to instantiate and implement a sub-class

Syntax:

public **abstract** class Calendar **extends** Object
implements Serializable, Cloneable, Comparable<Calendar>

Example:

```
import java.util.Calendar;
public class CalendarExample1 {
public static void main(String[] args) {

Calendar calendar = Calendar.getInstance();
 System.out.println("The current date is : " + calendar.getTime());

 calendar.add(Calendar.DATE, -15);
  System.out.println("15 days ago: " + calendar.getTime());

 calendar.add(Calendar.MONTH, 4);
  System.out.println("4 months later: " + calendar.getTime());

 calendar.add(Calendar.YEAR, 2);
   System.out.println("2 years later: " + calendar.getTime());
   }
}
```

Knowledge Check

Q1. When does method overloading is determined?

- At run time
- At compile time
- At coding time
- At execution time

Q2. What is it called where child object gets killed if parent object is killed?

- Aggregation
- Composition
- Encapsulation
- Association

Q3. Method overriding is combination of inheritance and polymorphism?

- True
- False

Q4. Which of these methods of class String is used to compare two String objects for their equality?

- equals()
- Equals()
- isequal()
- None of the above

Knowledge Check

Q5. Which of these packages contains abstract keyword?

- java.lang
- java.util
- java.io
- java.system

Q6. Static members are not inherited to subclass

- True
- False

Q7. Which of these methods of class StringBuffer is used to concatenate the string representation to the end of invoking string?

- concat()
- append()
- join()
- concatenate()

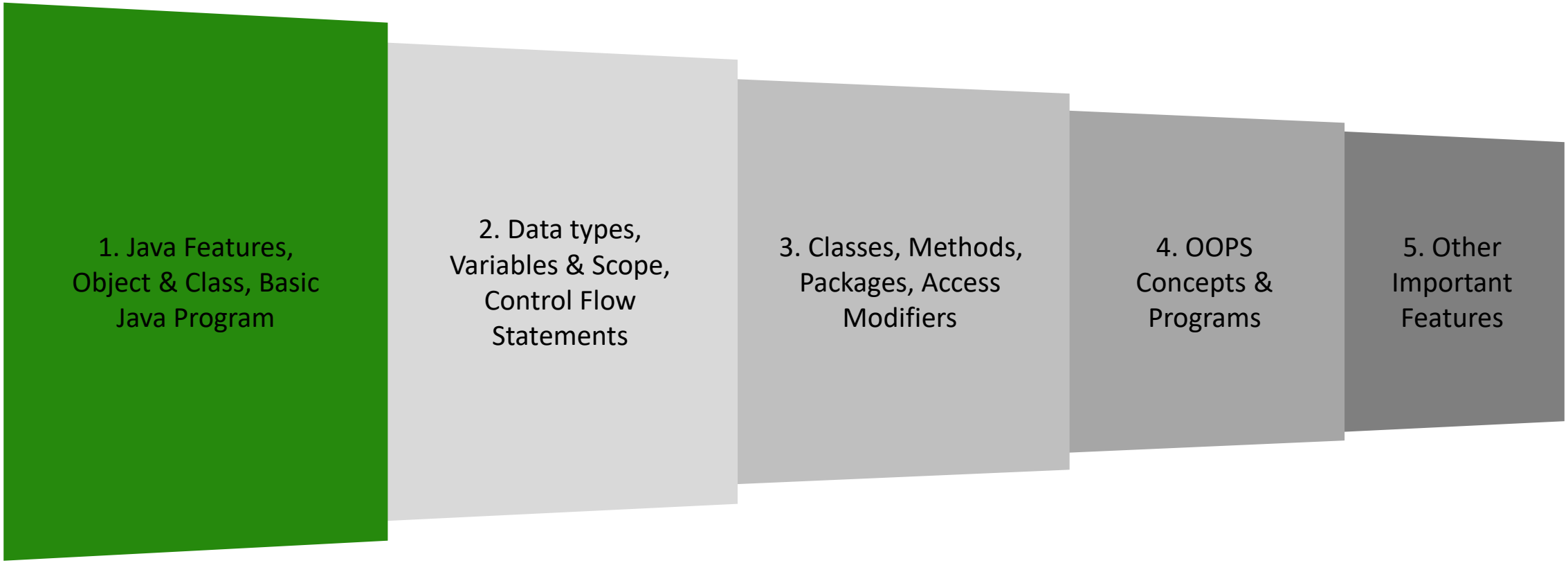
Q8. Which of these keywords cannot be used for a class which has been declared final?

- abstract
- extends
- Both
- None of the above

Recap

Lets summarize Java Basics

Glimpse of Important points



Any Questions?

Thank You