Core Java

- static keyword
  - Static block
  - Static import
- Singleton class
- Association (has-a relation)
- Inheritance (is-a relation)

static keyword

Static block

- Like Object/Instance initializer block, a class can have any number of static initialization blocks, and they can appear anywhere in the class body.
- Static initialization blocks are executed in the order their declaration in the class.
- A static block is executed only once when a class is loaded in JVM.
- Example:

```
class Program {
    static int field1 = 10;
    static int field2;
    static int field3;
    static final int field4;

    static {
        // static fields initialization
        field2 = 20;
        field3 = 30;
    }

    static {
        // initialization code
        field4 = 40;
    }
}
```

Static import

- To access static members of a class in the same class, the "ClassName." is optional.
- To access static members of another class, the "ClassName." is mandetory.
- If need to access static members of other class frequently, use "import static" so that we can access static members of other class direcly (without ClassName.).

```
import static java.lang.Math.pow;
import static java.lang.Math.sqrt;
import static java.lang.Math.abs;

class Program {
    public static double distance(int x1, int y1, int x2, int y2) {
        int dx = abs(x1 - x2), dy = abs(y1 - y2);
        double dist = sqrt( pow(dx, 2) + pow(dy, 2) );
        return dist;
    }
}
```

Singleton class

- Design patterns are standard solutions to the well-known problems.
- Singleton is a design pattern.
- It enables access to an object throughout the application source code.
- Singleton class is a class whose single object is created throughout the application.
- To make a singleton class in Java
    - step 1: Write a class with desired fields and methods.
    - step 2: Make constructor(s) private.
    - step 3: Add a private static field to hold instance of the class.
    - step 4: Initialize the field to single object using static field initializer or static block.
    - step 5: Add a public static method to return the object.
- Code:

```
public class Singleton {
    // fields and methods
    // since ctor is declared private, object of the class cannot be created
outside the class.
    private Singleton() {
        // initialization code
    }
    // holds reference of "the" created object.
    private static Singleton obj;
    static {
        // as static block is executed once, only one object is created
        obj = new Singleton();
    }
    // static getter method so that users can access the object
    public static Singleton getInstance() {
        return obj;
    }
}
```

```
class Program {
    public static void testMethod() {
        Singleton obj2 = Singleton.getInstance();
        // ...
    }

    public static void main(String[] args) {
        Singleton obj1 = Singleton.getInstance();
        // ...
    }
}
```

Association

- If "has-a" relationship exist between the types, then use association.
- To implement association, we should declare instance/collection of inner class as a field inside another class.
- There are two types of associations
    - Composition
    - Aggregation
- Example 1:

```
public class Engine {
    // ...
}
```

```
public class Person {
    private String name;
    private int age;
    // ...
}
```

```
public class Car {
    private Engine engine;
    private Person driver;
    // ...
}
```

- Example 2:

```
public class Wall {
    // ...
}
```

```
public class Person {
    // ...
}
```

```
public class Classroom {
    private Wall[] walls = new Wall[4];
    private ArrayList<Person> students = new ArrayList<>();
    // ...
}
```

Composition

- Represents part-of relation i.e. tight coupling between the objects.
- The inner object is essential part of outer object.
    - Engine is part of Car.
    - Wall is part of Car.

Association

- Represents has-a relation i.e. loose coupling between the objects.
- The inner object can be added, removed, or replaced easily in outer object.
    - Car has a Driver.
    - Company has Employees.

Inheritance

- If "is-a"/"kind-of" relationship exist between the types, then use inheritance.
- Inheritance is process -- generalization to specialization.
- All members of parent class are inherited to the child class.
- Example:
    - Manager is a Employee
    - Mango is a Fruit
    - Triangle is a Shape
- In Java, inheritance is done using extends keyword.

```
class SubClass extends SuperClass {
    // ...
}
```

- Java doesn't support multiple implementation inheritance i.e. a class cannot be inherited from multiple super-classes.
- However Java does support multiple interface inheritance i.e. a class can be inherited from multiple super interfaces.

super keyword

- In sub-class, super-class members are referred using "super" keyword.
- Calling super class constructor
  - By default, when sub-class object is created, first super-class constructor (param-less) is executed and then sub-class constructor is executed.
  - "super" keyword is used to explicitly call super-class constructor.

```
class Person {
    // ...
    public Person(String name, int age) {
        // ...
    }
}
class Student extends Person {
    // ...
    public Student(String name, int age, int roll, double marks) {
        super(name, age); // calls parameterized ctor of super class -- must
be first line only
        // ...
    }
}
```

- Accessing super class members
  - Super class members (non-private) are accessible in sub-class directly or using "this" reference. These members can also be accessed using "super" keyword.
  - However, if sub-class method signature is same as super-class signature, it hides/shadows method of the super class i.e. super-class method is not directly visible in sub-class.
  - The "super" keyword is mandetory for accessing such hidden members of the super-class.

```
class Person {
    // ...
    public String getName() {
        // ...
    }
    public int getAge() {
        // ...
    }
    public void display() {
        // display name and age
    }
```

```
}
class Student extends Person {
    // ...
    public void display() {
        System.out.println(this.getName()); // getName() is inherited from
super-class
        System.out.println(getAge()); // getAge() is inherited from super-
class
        super.display(); // Person.display() is hidden due to
Student.display()
            // must use super keyword to call hidden method of super class.
        // display roll and marks
    }
}
```