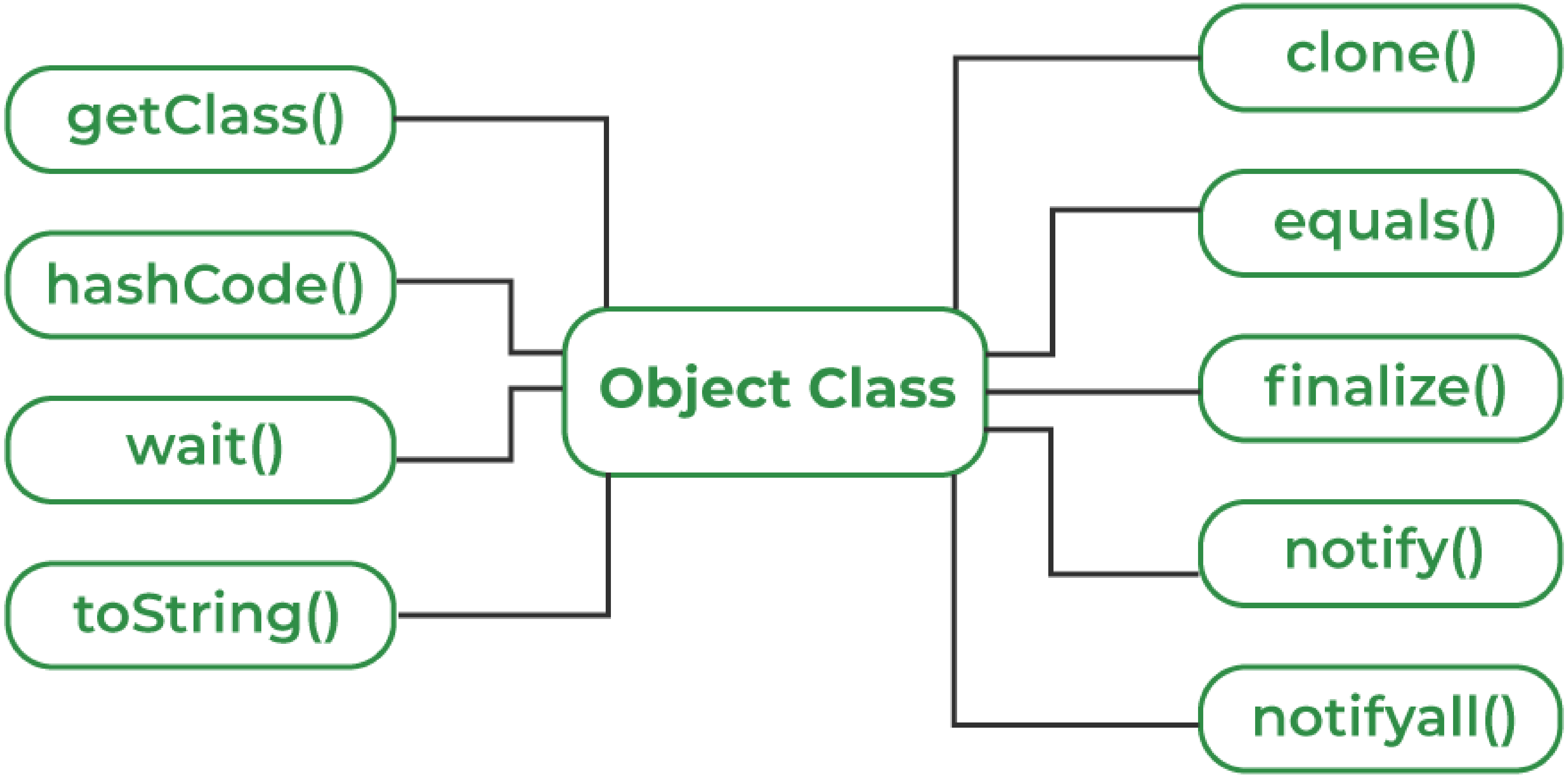
Object Class

Object class in Java

- The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.
- The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as upcasting.
- Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee,Student etc, we can use Object class reference to refer that object

- clone(): Creates and returns a copy of this object.
- equals(Object obj): Indicates whether some other object is "equal to" this one.
- finalize(): Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
- getClass(): Gets the runtime class of this Object.
- hashCode(): Returns a hash code value for the object.
- toString(): Returns a string representation of the object.
- notify(), notifyAll(), and wait(): Are part of the mechanism that Java uses for thread synchronization.


Object Class
getClass()
hashCode()
wait()
toString()
clone()
equals()
finalize()
notify()
notifyall()

```
public class Car {
   private String make;
   private String model;
   public Car(String make, String model) {
      this.make = make;
      this.model = model;
   }
   // Override toString() method from Object class
   @Override
   public String toString() {
      return "Car{" +
             "make='" + make + '\'' +
             ", model='" + model + '\'' +
             '}';
   }
   public static void main(String[] args) {
      Car myCar = new Car("Toyota", "Corolla");
      System.out.println(myCar.toString());  // Explicitly calling toString()
      System.out.println(myCar);             // Implicitly calling toString()
   } }
```

- toString()

```
Car{make='Toyota', model='Corolla'}
Car{make='Toyota', model='Corolla'}
```

```java
class Coordinate {
    int x, y;
    Coordinate(int x, int y) {
        this.x = x;
        this.y = y;
    }
    // Override toString method
    @Override
    public String toString() {
        return "Coordinate: [" + x + ", " + y + "]";
    }
    public static void main(String[] args) {
        Coordinate point = new Coordinate(5, 10);
        System.out.println(point);
    }
}
```

```
Coordinate: [5, 10]
```

```java
class Box {
    int width, height, depth;
    // Constructor for Box class
    Box(int w, int h, int d) {
        width = w;
        height = h;
        depth = d;
    }
    // Overriding the toString method
    @Override
    public String toString() {
        return "Box(" + width + ", " + height + ", " + depth + ")";
    }
    public static void main(String[] args) {
        Box myBox = new Box(10, 20, 30);
        System.out.println(myBox);
    }
}
```

```
Box(10, 20, 30)
```

```
// Here we have a class Animal, which is like a general type of object.
class Animal {
   void whoAmI() {
      System.out.println("I am an animal");
   } }
// Here is a more specific type of Animal, let's say a Dog.
class Dog extends Animal {
   void whoAmI() {
      System.out.println("I am a dog");
   }}
public class Main {
   public static void main(String[] args) {
      // Here we create a Dog, but we refer to it as an Animal.
      // This is upcasting. We're treating the Dog just as a general Animal.
      Animal myDog = new Dog();
      // When we call the whoAmI method, it says "I am a dog" because
      // even though we're treating it as an Animal, it's actually a Dog.
      myDog.whoAmI();
   } }
```

UPCASTING

Upcasting is the process of treating a subclass object as an instance of its superclass

```
class Fruit {
   void show() {
      System.out.println("This is a fruit.");
   } }
class Apple extends Fruit {
   void show() {
      System.out.println("This is an apple.");
   } }
class Banana extends Fruit {
   void show() {
      System.out.println("This is a banana.");
   } }
public class Main {
   public static void main(String[] args) {
      Object fruitBox1 = new Apple(); // Apple is treated as an Object
      Object fruitBox2 = new Banana(); // Banana is treated as an Object
      // Despite being specific fruits, we refer to them using the Object class
      System.out.println(fruitBox1.getClass().getName()); // Outputs "Apple"
      System.out.println(fruitBox2.getClass().getName()); // Outputs "Banana"
   } }
```

- getClass()

```
public class GetClassExample {

   public static void main(String[] args) {
      // Create an instance of String
      String myString = "Hello, World!";

      // Use getClass() to print out the class of the object
      System.out.println("The class of myString is: " +
myString.getClass().getName());
   }
}
```

```
The class of myString is: java.lang.String
```

Inner Class in Java

Member Inner Class
Anonymous Inner Classes
Local Inner Classes

Java Inner Classes (Nested Classes)

Java inner class or nested class is a class that is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place to be more readable and maintainable.

Additionally, it can access all the members of the outer class, including private data members and methods.

Syntax of Inner class

```
class Java_Outer_class
{
 //code
 class Java_Inner_class
{
  //code
 }  }
```

- ❑ **Non-static nested class (inner class)**
  - **Member inner class**
  - **Anonymous inner class**
  - **Local inner class**

- ❑ **Static nested class**

Member Inner Class: A class created within class and outside method.

```
class OuterClass
 {
   class InnerClass
      {
      // methods and fields - can access members of the
OuterClass.
      }
}
```

```
public class Circle {
    private double radius;
    private Point center; // Instance of the inner class
    public Circle(double radius, double x, double y) {
        this.radius = radius;
        this.center = new Point(x, y);
    }
    public void displayCenter() {
        System.out.println("Circle center at (" + center.x + ", " + center.y + ")");
    }
    // Member Inner Class
    class Point {
        double x, y;
        public Point(double x, double y) {
            this.x = x;
            this.y = y;
        }
    }
    public static void main(String[] args) {
        Circle circle = new Circle(5.0, 1.0, 2.0);
        circle.displayCenter();
    }
}
```

Local Inner Class: A class defined within a block, typically a method block.

```
class OuterClass
 {
   void myMethod()
   {
      class LocalInnerClass
     {
        // can only be used within the block where it is defined.
      }
   }
}
```

```
public class Greeting {
   public void greetInLanguage(String language) {
      // Local Inner Class
      class Greeter {
         void greet() {
            switch (language) {
               case "English":
                  System.out.println("Hello!");
                  break;
               case "Spanish":
                  System.out.println("Hola!");
                  break;
               case "French":
                  System.out.println("Bonjour!");
                  break;
               default:
                  System.out.println("Language not supported.");
            }
         }    }
      Greeter greeter = new Greeter();
      greeter.greet();
   }
   public static void main(String[] args) {
      Greeting greeting = new Greeting();
      greeting.greetInLanguage("French");} }
```

Static Nested Class: A static class created inside a class.

```
class OuterClass
 {
   static class NestedStaticClass
 {
      // can exist independently of an instance of
OuterClass
}
}
```

```
public class Computer {
   private String name;
   private Processor processor;
   public Computer(String name, String processorType, double processorSpeed) {
      this.name = name;
      this.processor = new Processor(processorType, processorSpeed);
   }
   public void displayComputerDetails() {
      System.out.println("Computer Name: " + name);
      System.out.println("Processor Type: " + processor.getType());
      System.out.println("Processor Speed: " + processor.getSpeed() + "GHz");
   }
   // Static Nested Class
   static class Processor {
      private String type;
      private double speed;
      public Processor(String type, double speed) {
         this.type = type;
         this.speed = speAed;
      }
      public String getType() {
         return type;
      }
      public double getSpeed() {
         return speed;
      }
   }
   public static void main(String[] args)
 {
      Computer myComputer = new Computer("MyComputer", "Intel i7", 3.5);
      myComputer.displayComputerDetails();
   }
```

Java Garbage Collection
?

Java Garbage Collection

- In java, garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically.
- In other words, it is a way to destroy the unused objects.
- To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

How can an object be unreferenced?

01 By nulling the reference

02 By assigning a reference to another

03 By anonymous object etc.

Nulling the Reference

When you set a reference to null, you are explicitly stating that the reference no longer points to any object. After this point, the object it previously pointed to is eligible for garbage collection if there are no other references to it.

```
public class Example
{
   public static void main(String[] args)
 {
       String str = new String("Example string");
       str = null; // Now the String object is eligible for garbage
collection.
   }
}
```

Assigning a Reference to Another

If you have a reference variable that points to an object, and you assign it to another object, the first object becomes unreferenced if no other references exist.

```
public class Example
{
   public static void main(String[] args)
 {
      StringBuilder builder = new StringBuilder("Initial");
      StringBuilder anotherBuilder = new StringBuilder("Another");

      builder = anotherBuilder; // The StringBuilder object containing
"Initial" is now eligible for garbage collection.
   }
}
```

Anonymous Object

An anonymous object is an object that is instantiated but not stored in a reference variable. After the line of code that creates and uses the anonymous object is executed, the object is eligible for garbage collection because it cannot be reached by any live threads.

```
public class Example {
   public static void main(String[] args) {
      new String("This is an anonymous object"); // After this statement,
the String object is eligible for garbage collection.

      // Here we use an anonymous object to perform an operation.
      System.out.println(new StringBuilder("Anonymous
StringBuilder").reverse());
   }
}
```

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

1.public static void gc(){}

Finalize Method.

Java Object finalize() Method

Finalize() is the method of Object class. This method is called just before an object is garbage collected. finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks.

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

```
protected void finalize(){}
```

```
public class TestGarbage1{
 public void finalize()
{ System.out.println("object is garbage collected");}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();
 }
}
```

```
class FinalizationDemo {

   @Override
   protected void finalize() {
      System.out.println("Finalize method called.");
   }

   public static void main(String[] args) {
      FinalizationDemo obj = new FinalizationDemo();
      obj = null; // Make obj eligible for garbage collection

      // Suggesting garbage collection. Note that it is NOT guaranteed to run
immediately.
      System.gc();
      System.out.println("Garbage Collection is requested");
   }
}
```