

## 11-11-2025(case studies)

---

### Q1. Intergalactic Transport System

Code:

```
/** Represents a general spacecraft with a given fuel level. Defines abstract
 * launch behavior and a default landing procedure.
 *
 * @param fuelLevel
 *   The amount of fuel available in the spacecraft.
 */
abstract class Spacecraft(val fuelLevel: Int) {

    /** Initiates the spacecraft's launch sequence. */
    def launch(): Unit

    /** Performs a standard landing sequence. Default implementation prints a
     * generic landing message.
     */
    def land(): Unit =
        println("Standard landing procedure initiated. Touchdown successful.")
}

/** Provides autonomous navigation capabilities for spacecraft. Includes a
 * default implementation for automatic navigation.
 */
trait Autopilot {

    /** Engages the autopilot navigation system. */
    def autoNavigate(): Unit = println("Autopilot engaged.")
}

/** A spacecraft designed for transporting cargo. It supports launch, landing,
 * and autopilot functionalities.
 *
 * @param fuelLevel
 *   Amount of fuel available.
 */
class CargoShip(fuelLevel: Int) extends Spacecraft(fuelLevel) with Autopilot {
```

```

/** Launch sequence specific to cargo spacecraft. */
override def launch(): Unit =
    println("CargoShip firing main thrusters for liftoff!")

/** Performs a custom landing operation. Marked as final to prevent further
    * overriding.
    */
final override def land(): Unit =
    println("CargoShip performing automated landing and cargo unloading.")
}

/** A spacecraft designed to carry passengers. Extends Spacecraft and includes
    * autopilot capabilities.
    *
    * @param fuelLevel
    *   Amount of fuel available.
    */
class PassengerShip(fuelLevel: Int)
    extends Spacecraft(fuelLevel)
    with Autopilot {

    /** Launch sequence specific to passenger spacecraft with safety checks. */
    override def launch(): Unit =
        println("PassengerShip commencing passenger safety checks and takeoff!")

    /** Custom landing procedure including passenger safety protocols. */
    override def land(): Unit =
        println("PassengerShip land: Safety protocols for all passengers.")
}

/** A luxury variant of the PassengerShip. Includes entertainment features for
    * passengers.
    *
    * This class is final and cannot be extended further.
    *
    * @param fuelLevel
    *   Amount of fuel available.
    */
final class LuxuryCruiser(fuelLevel: Int) extends PassengerShip(fuelLevel) {

    /** Plays onboard entertainment for luxury passengers. */

```

```

def playEntertainment(): Unit =
    println("Welcome to the intergalactic theater: Enjoy movies and music!")
}

/** Demonstrates spacecraft operations using cargo, passenger, and luxury ships.
  */
object IntergalacticTransportSystem extends App {

    /** Instance of a cargo spacecraft with fuel level 100. */
    val cargoShip: CargoShip = CargoShip(100)

    /** Instance of a passenger spacecraft with fuel level 200. */
    val passengerShip: PassengerShip = PassengerShip(200)

    /** Instance of a luxury spacecraft with fuel level 300. */
    val luxuryCruiser: LuxuryCruiser = LuxuryCruiser(300)

    cargoShip.launch()
    cargoShip.land()
    cargoShip.autoNavigate()
    println(s"Cargo Ship Fuel Level: ${cargoShip.fuelLevel}\n")

    passengerShip.launch()
    passengerShip.land()
    passengerShip.autoNavigate()
    println(s"Passenger Ship Fuel Level: ${passengerShip.fuelLevel}\n")

    luxuryCruiser.launch()
    luxuryCruiser.land()
    luxuryCruiser.autoNavigate()
    luxuryCruiser.playEntertainment()
    println(s"Luxury Cruiser Fuel Level: ${luxuryCruiser.fuelLevel}\n")
}

```

Output:

```
PROBLEMS 52 OUTPUT DEBUG CONSOLE TERMINAL PORTS
rac@PTPMRT07 ScalaTraining % scala 11-11-2025/IntergalacticTransportSystem.scala
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe:objectFieldOffset has been called by scala.runtime.LazyVals$ (file:/opt/homebrew/Cellar/sc
ala/3.7.3/libexec/libexec/scala-cli.jar)
WARNING: Please consider reporting this to the maintainers of class scala.runtime.LazyVals$
WARNING: sun.misc.Unsafe:objectFieldOffset will be removed in a future release
Compiling project (Scala 3.7.3, JVM (25))
Compiled project (Scala 3.7.3, JVM (25))
CargoShip firing main thrusters for liftoff!
CargoShip performing automated landing and cargo unloading.
Autopilot engaged.
Cargo Ship Fuel Level: 100

PassengerShip commencing passenger safety checks and takeoff!
PassengerShip land: Safety protocols for all passengers.
Autopilot engaged.
Passenger Ship Fuel Level: 200

PassengerShip commencing passenger safety checks and takeoff!
PassengerShip land: Safety protocols for all passengers.
Autopilot engaged.
Welcome to the intergalactic theater: Enjoy movies and music!
Luxury Cruiser Fuel Level: 300
rac@PTPMRT07 ScalaTraining %
```

## Q2. Smart Home Automation System

Code:

```
package SmartHomeAutomationSystem

/** Represents a general smart home device. Provides basic operations such as
 * turning on, turning off, and checking status.
 */
trait Device {

  /** Turns the device on. */
  def turnOn(): Unit

  /** Turns the device off. */
  def turnOff(): Unit

  /** Prints the current operational status of the device. Default
   * implementation indicates that the device is operational.
   */
  def status(): Unit = println("Device is operational")
}

/** Adds network connectivity support to a device. Includes methods to connect
 * and disconnect from the network.
 */
```

```

trait Connectivity {

  /** Connects the device to the smart home network. */
  def connect(): Unit = println("Device connected to network")

  /** Disconnects the device from the network. */
  def disconnect(): Unit = print("Device disconnected")
}

/** Adds voice control functionality to a device. Overrides turnOn and turnOff
  * to simulate voice-activated actions.
  */
trait VoiceControl {

  /** Turns the device on via voice command. */
  def turnOn(): Unit =
    println("Voice control: Device turned on")

  /** Turns the device off via voice command. */
  def turnOff(): Unit =
    println("Voice control: Device turned off")
}

/** Enhances a device with energy-saving capabilities. Overrides the shutdown
  * behavior to conserve energy.
  */
trait EnergySaver extends Device {

  /** Activates the device's energy-saving mode. */
  def activateEnergySaver(): Unit =
    println("Energy saver mode activated")

  /** Custom shutdown behavior designed to reduce power usage. Overrides the
    * default Device turnOff method.
    */
  override def turnOff(): Unit =
    println("Device powered down to save energy")
}

/** A smart light device equipped with connectivity and energy-saving features.
  * Provides custom behavior for turning the light on.
  */

```

```

class SmartLight extends Device with Connectivity with EnergySaver {

  /** Turns the smart light on with a bright illumination message. */
  override def turnOn(): Unit =
    println("SmartLight is shining brightly!")
}

/** A smart thermostat device equipped with connectivity and energy-saving
 * features. Provides custom behavior for turning the thermostat on and off.
 */
class SmartThermostat extends Device with Connectivity with EnergySaver {

  /** Activates the thermostat heating mechanism. */
  override def turnOn(): Unit =
    println("SmartThermostat warming up!")

  /** Custom safe shutdown sequence for the thermostat. */
  override def turnOff(): Unit =
    println("SmartThermostat shutting down safely.")
}

/** Demonstrates smart device operations and trait-based feature composition in
 * a Smart Home Automation System.
 */
object SmartHomeAutomationSystem extends App {

  /** Instance of a smart light device. */
  val smartLight = new SmartLight

  /** Instance of a smart thermostat device. */
  val smartThermostat = new SmartThermostat

  smartLight.turnOn()
  smartLight.connect()
  smartLight.activateEnergySaver()
  smartLight.turnOff()
  smartLight.status()

  println("_____")

  smartThermostat.turnOn()
  smartThermostat.connect()

```

```

smartThermostat.activateEnergySaver()
smartThermostat.turnOff()
smartThermostat.status()

println("_____")

/** A smart light enhanced with voice control. Demonstrates resolving method
 * conflicts using super with trait qualifiers.
 */
val voiceControl = new SmartLight with VoiceControl {
  override def turnOn(): Unit =
    super[VoiceControl].turnOn()

  override def turnOff(): Unit =
    super[VoiceControl].turnOff()
}

voiceControl.turnOn()
voiceControl.turnOff()
}

```

Output:

```

rac@PTPMRT07 ScalaTraining % scala 11-11-2025/IntergalacticTransportSystem.scala

● rac@PTPMRT07 ScalaTraining % scala 11-11-2025/SmartHomeAutomationSystem.scala
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::objectFieldOffset has been called by scala.runtime.LazyVals$ (file:/opt/homebrew/Cellar/sc
ala/3.7.3/libexec/libexec/scala-cli.jar)
WARNING: Please consider reporting this to the maintainers of class scala.runtime.LazyVals$
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
Compiling project (Scala 3.7.3, JVM (25))
Compiled project (Scala 3.7.3, JVM (25))
SmartLight is shining brightly!
Device connected to network
Energy saver mode activated
Device powered down to save energy
Device is operational

-----
SmartThermostat warming up!
Device connected to network
Energy saver mode activated
SmartThermostat shutting down safely.
Device is operational

-----
Voice control: Device turned on
Voice control: Device turned off
○ rac@PTPMRT07 ScalaTraining %

```

### Q3. Personalized Robot Assistant System

Code:

```
/** Represents a general robot with basic lifecycle operations. Includes methods
 * to start, shut down, and check operational status.
 */
trait Robot {

  /** Starts the robot's core systems. */
  def start(): Unit

  /** Shuts down the robot. */
  def shutdown(): Unit

  /** Prints the current operational status of the robot. Default implementation
   * indicates the robot is operational.
   */
  def status(): Unit = println("Robot is operational")
}

/** Provides speech capabilities for a robot. Enables the robot to speak
 * messages aloud.
 */
trait SpeechModule {

  /** Generates a speech output with the given message.
   *
   * @param message
   *   The message to be spoken by the robot.
   */
  def speak(message: String): Unit =
    println(s"Robot says: $message")
}

/** Adds movement capabilities to a robot. Supports both forward and backward
 * movement.
 */
trait MovementModule {

  /** Moves the robot forward. */
  def moveForward(): Unit = println("Moving forward")
}
```

```

/** Moves the robot backward. */
def moveBackward(): Unit = println("Moving backward")
}

/** Adds energy-saving behavior to a robot. Overrides the shutdown process to
 * conserve power.
 */
trait EnergySaver extends Robot {

  /** Activates energy-saving mode. */
  def activateEnergySaver(): Unit = println("Energy saver mode activated")

  /** Custom shutdown behavior designed to conserve energy. Overrides the
   * default Robot shutdown method.
   */
  override def shutdown(): Unit =
    println("Robot shutting down to save energy")
}

/** A simple implementation of a robot. Provides basic start and shutdown
 * actions.
 */
class BasicRobot extends Robot {

  /** Starts the basic robot systems. */
  override def start(): Unit =
    println("BasicRobot starting up")

  /** Shuts down the basic robot systems. */
  override def shutdown(): Unit =
    println("BasicRobot shutting down")
}

/** Demonstrates various robot configurations using behavior-mixing traits.
 */
object PersonalizedRobotAssistantSystem extends App {

  /** Robot with speech and movement capabilities. */
  val myRobot = new BasicRobot with SpeechModule with MovementModule
  myRobot.start()
  myRobot.status()
  myRobot.speak("Hello!")
}

```

```

myRobot.moveForward()
myRobot.shutdown()

println("-----")

/** Robot with speech, movement, and energy-saving abilities. */
val energyRobot = new BasicRobot
  with SpeechModule
  with MovementModule
  with EnergySaver
energyRobot.start()
energyRobot.speak("Saving energy now")
energyRobot.moveBackward()
energyRobot.activateEnergySaver()
energyRobot.shutdown()

println("-----")

/** Robot with mixed-in traits in a different order to illustrate Scala's
    * trait linearization and method resolution.
    */
val reversedRobot = new BasicRobot
  with SpeechModule
  with EnergySaver
  with MovementModule
reversedRobot.start()
reversedRobot.speak("Testing method resolution")
reversedRobot.moveForward()
reversedRobot.activateEnergySaver()
reversedRobot.shutdown()
}

```

Output:

```
rac@PTPMRT07 ScalaTraining % scala 11-11-2025/PersonalizedRobotAssistantSystem.scala
WARNING: sun.misc.Unsafe::objectFieldOffset has been called by scala.runtime.LazyVals$ (file:/opt/homebrew/Cellar/sc
ala/3.7.3/libexec/libexec/scala-cli.jar)
WARNING: Please consider reporting this to the maintainers of class scala.runtime.LazyVals$
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
Compiling project (Scala 3.7.3, JVM (25))
Compiled project (Scala 3.7.3, JVM (25))
BasicRobot starting up
Robot is operational
Robot says: Hello!
Moving forward
BasicRobot shutting down

-----
BasicRobot starting up
Robot says: Saving energy now
Moving backward
Energy saver mode activated
Robot shutting down to save energy

-----
BasicRobot starting up
Robot says: Testing method resolution
Moving forward
Energy saver mode activated
Robot shutting down to save energy
rac@PTPMRT07 ScalaTraining %
```

#### Q4. Intergalactic drone

Code:

```
/** Represents the core functionality of a drone. Provides methods to activate,
 * deactivate, and check the status of the drone.
 */
trait Drone {

  /** Activates the drone */
  def activate(): Unit

  /** Deactivates the drone */
  def deactivate(): Unit

  /** Prints the current operational status of the drone. Default implementation
   * prints "Drone status: operational".
   */
  def status(): Unit = println("Drone status: operational")
}

/** Provides navigation capabilities for a drone. Includes the ability to fly to
 * a specified destination.
 */
/** Extends [[Drone]] and overrides the deactivate method.
 */
```

```

trait NavigationModule extends Drone {

  /** Navigates the drone to the given destination.
    *
    * @param destination
    *   The target location to fly to.
    */
  def flyTo(destination: String): Unit =
    println(s"Flying to $destination")

  /** Deactivates navigation-specific systems. */
  override def deactivate(): Unit =
    println("Navigation systems shutting down")
}

/** Adds defensive capabilities to a drone. Includes shield activation.
  *
  * Extends [[Drone]] and overrides the deactivate method.
  */
trait DefenseModule extends Drone {

  /** Activates the drone's defense shields. */
  def activateShields(): Unit = println("Shields activated")

  /** Deactivates defense-related systems. */
  override def deactivate(): Unit =
    println("Defense systems deactivated")
}

/** Adds communication capabilities to a drone. Includes the ability to send
  * messages.
  *
  * Extends [[Drone]] and overrides the deactivate method.
  */
trait CommunicationModule extends Drone {

  /** Sends a communication message.
    *
    * @param msg
    *   The message to be transmitted.
    */
  def sendMessage(msg: String): Unit =

```

```

    println(s"Sending message: $msg")

    /** Shuts down the communication module. */
    override def deactivate(): Unit =
        println("Communication module shutting down")
}

/** A simple drone implementation with basic activate and deactivate behavior.
    */
class BasicDrone extends Drone {

    /** Activates the basic drone. */
    override def activate(): Unit =
        print("BasicDrone activated")

    /** Deactivates the basic drone. */
    override def deactivate(): Unit =
        print("BasicDrone deactivated")
}

/** Demonstrates different combinations of drone modules using Scala traits
    * mixed into a basic drone.
    */
object IntergalacticDrone extends App {

    /** Drone with navigation and defense capabilities. */
    val drone1 = new BasicDrone with NavigationModule with DefenseModule
    drone1.activate()
    drone1.status()
    drone1.flyTo("Mars")
    drone1.activateShields()
    drone1.deactivate()

    println("_____")

    /** Drone with communication and navigation modules. */
    val drone2 = new BasicDrone with CommunicationModule with NavigationModule
    drone2.activate()
    drone2.status()
    drone2.sendMessage("Hello, Earth!")
    drone2.flyTo("Venus")
    drone2.deactivate()

```

```
println("_____")

/** Drone with defense, communication, and navigation capabilities. */
val drone3 = new BasicDrone
  with DefenseModule
  with CommunicationModule
  with NavigationModule
drone3.activate()
drone3.status()
drone3.activateShields()
drone3.sendMessage("Mission Complete")
drone3.flyTo("Jupiter")
drone3.deactivate()
}
```

Output:

```
rac@PTPMRT07 ScalaTraining % scala 11-11-2025/PersonalizedRobotAssistantSystem.scala
rac@PTPMRT07 ScalaTraining % scala 11-11-2025/IntergalacticDrone.scala
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::objectFieldOffset has been called by scala.runtime.LazyVals$ (file:/opt/homebrew/Cellar/sc
ala/3.7.3/libexec/libexec/scala-cli.jar)
WARNING: Please consider reporting this to the maintainers of class scala.runtime.LazyVals$
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
Compiling project (Scala 3.7.3, JVM (25))
Compiled project (Scala 3.7.3, JVM (25))
BasicDrone activatedDrone status: operational
Flying to Mars
Shields activated
Defense systems deactivated

-----
BasicDrone activatedDrone status: operational
Sending message: Hello, Earth!
Flying to Venus
Navigation systems shutting down

-----
BasicDrone activatedDrone status: operational
Shields activated
Sending message: Mission Complete
Flying to Jupiter
Navigation systems shutting down
rac@PTPMRT07 ScalaTraining %
```

## Q5. Smart Library Management System

Code:

SmartLibraryManagmentSystem.scala

```
import library.items._
```

```

import library.users._
import library.operations.LibraryOps._

/**
 * Entry point for the Smart Library Management System.
 *
 * Demonstrates various library operations such as borrowing books,
 * magazines, and DVDs, as well as working with members and item descriptions.
 *
 * This object relies on:
 * - `library.items` for item types like Book, Magazine, DVD
 * - `library.users` for user types such as Member
 * - `library.operations.LibraryOps` for borrowing logic and item utilities
 */
object SmartLibraryManagementSystem extends App {

  /** Example: Borrowing a book using an implicit default member. */
  borrow("The Scala Language")

  /** Creates a magazine instance and borrows it. */
  val mag = Magazine("Tech Monthly")
  borrow(mag)

  /** Creates a DVD instance and borrows it. */
  val dvd = DVD("Interstellar")

  /** Creates a library member explicitly. */
  val member: Member = new Member("Alice")

  /** Borrow a DVD using the default implicit member from scope. */
  borrow(dvd)

  /** Borrow a book using an explicitly provided Member instance. */
  borrow(Book("Harry Potter"))(using member)

  /** Prints a description of a DVD item. */
  println(itemDescription(dvd))

  /** Prints a description for a book title provided as a String. */
  println(itemDescription("Functional Programming with Scala"))
}

```

## ItemType.scala

```
package library.items
```

```
/** Represents a general type of library item.
 *
 * All items in the library—such as books, magazines, and DVDs—extend this
 * trait. Each item type must define a `title` describing the item.
 */
sealed trait ItemType {

  /** The title or name of the library item. */
  def title: String
}

/** Represents a book available in the library.
 *
 * @param title
 *   The title of the book.
 */
case class Book(title: String) extends ItemType

/** Represents a magazine available in the library.
 *
 * @param title
 *   The title of the magazine.
 */
case class Magazine(title: String) extends ItemType

/** Represents a DVD available in the library.
 *
 * @param title
 *   The title of the DVD.
 */
case class DVD(title: String) extends ItemType
```

## LibraryOps.scala

```
package library.operations
```

```
import library.items._
import library.users._
```

```

/** Contains core operations used in the Smart Library Management System.
 *
 * Provides functionality for:
 *   - Borrowing items
 *   - Describing library items
 *   - Converting Strings to Book instances
 *
 * Also defines an implicit default member used when no explicit member is
 * provided.
 */
object LibraryOps {

  /** An implicit default library member used automatically when no member is
    * passed to borrowing operations.
    */
  implicit val defaultMember: Member = new Member("Default Member")

  /** Allows a library member to borrow a specific item.
    *
    * This method requires an implicit [[library.users.Member]]. If none is
    * provided, the `defaultMember` is used.
    *
    * @param item
    *   The library item to be borrowed.
    * @param member
    *   The member borrowing the item (implicit).
    */
  def borrow(item: ItemType)(implicit member: Member): Unit =
    member.borrowItem(item)

  /** Produces a readable description of a given library item.
    *
    * @param item
    *   The item whose description is needed.
    * @return
    *   A string describing the item based on its type.
    */
  def itemDescription(item: ItemType): String = item match {
    case Book(title)      => s"Book: '$title'"
    case Magazine(title) => s"Magazine: '$title'"
    case DVD(title)       => s"DVD: '$title'"
  }
}

```

```

}

/** Implicit conversion from a `String` to a `Book`.
 *
 * This enables borrowing or describing items by simply providing a title
 * string—for example: `borrow("The Scala Language")`.
 *
 * @param title
 *   The title of the book.
 * @return
 *   A `Book` instance with the given title.
 */
implicit def stringToBook(title: String): Book = Book(title)
}

```

## Member.scala:

```

package library.users

import library.items.ItemType

/** Represents a library member who can borrow library items.
 *
 * @param name
 *   The name of the member.
 */
class Member(val name: String) {

  /** Allows the member to borrow a specified library item.
   *
   * Prints a message indicating which item the member has borrowed.
   *
   * @param item
   *   The library item being borrowed.
   */
  def borrowItem(item: ItemType): Unit = {
    println(s"$name borrowed '${item.title}')"
  }
}

```

Output:

```
PROBLEMS 52 OUTPUT DEBUG CONSOLE TERMINAL PORTS
rac@PTPMRT07 ScalaTraining % scala -classpath out . SmartLibraryManagementSystem.scala --main-class SmartLibraryManagementSystem
[error] ./13-11-2025/scala-kafka-producer/src/main/scala/WebServer.scala:12:16
[error] type of implicit definition needs to be given explicitly
[error] implicit val system = ActorSystem(Behaviors.empty, "MyActorSystem")
[error] ^
Error compiling project (Scala 3.7.3, JVM (25))
Compilation failed
rac@PTPMRT07 ScalaTraining % scala -classpath out . SmartLibraryManagementSystem.scala --main-class SmartLibraryManagementSystem
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe:objectFieldOffset has been called by scala.runtime.LazyVals$ (file:/opt/homebrew/Cellar/scala/3.7.3/libexec/libexec/scala-cli.jar)
WARNING: Please consider reporting this to the maintainers of class scala.runtime.LazyVals$
WARNING: sun.misc.Unsafe:objectFieldOffset will be removed in a future release
Warning: setting /Users/rac/Desktop/ScalaTraining as the project root directory for this run.
Compiling project (Scala 3.7.3, JVM (25))
Compiled project (Scala 3.7.3, JVM (25))
Default Member borrowed 'The Scala Language'
Default Member borrowed 'Tech Monthly'
Default Member borrowed 'Interstellar'
Alice borrowed 'Harry Potter'
DVD: 'Interstellar'
Book: 'Functional Programming with Scala'
rac@PTPMRT07 ScalaTraining %
```

## Q6. Smart Traffic Management System

Code:

```
import java.io.FileInputStream
import java.sql._
import java.util.Properties
import scala.util.Using
import java.time.LocalDateTime
import scala.collection.mutable.ListBuffer
import scala.util.{Try, Success, Failure}

/** Represents a vehicle registered in the traffic system.
 *
 * @param vehicleId
 *   Optional unique ID assigned by the database.
 * @param licensePlate
 *   License plate number of the vehicle.
 * @param vehicleType
 *   Type of vehicle (e.g., car, bike, truck).
 * @param ownerName
 *   Full name of the vehicle owner.
 */
case class Vehicle(
  vehicleId: Option[Int] = None,
```

```

    licensePlate: String,
    vehicleType: String,
    ownerName: String
) {

  /** Returns a formatted string representation for table display. */
  override def toString: String =
    f"${vehicleId.getOrElse("")}%-10s | $licensePlate%-15s | $vehicleType%-12s |
$ownerName%-15s"
}

/** Represents a traffic signal in the system.
 *
 * @param signalId
 *   Optional unique ID generated by the database.
 * @param location
 *   Physical location of the traffic signal.
 * @param status
 *   Current signal state (green/yellow/red).
 */
case class TrafficSignal(
  signalId: Option[Int] = None,
  location: String,
  status: String
) {

  /** Returns a formatted string representation for table display. */
  override def toString: String = {
    f"${signalId.getOrElse("")}%-10s | $location%-20s | $status%-10s"
  }
}

/** Represents a traffic violation event recorded in the system.
 *
 * @param violationId
 *   Optional unique ID assigned by the database.
 * @param vehicleId
 *   ID of the involved vehicle.
 * @param signalId
 *   ID of the traffic signal where violation occurred.
 * @param violationType

```

```

*   Type of violation (e.g., speeding, signal jump).
*   @param timestamp
*   Time the violation occurred.
*/
case class Violation(
  violationId: Option[Int] = None,
  vehicleId: Int,
  signalId: Int,
  violationType: String,
  timestamp: LocalDateTime = LocalDateTime.now()
) {

  /** Returns a formatted string representation for table display. */
  override def toString: String = {
    f"${violationId.getOrElse("")}%-12s | $vehicleId%-10d | $signalId%-10d |
$violationType%-15s | $timestamp%-20s"
  }
}

/** Utility object responsible for handling database connections,
 *   initialization, and table creation.
 *
 *   * Loads configuration from a properties file, checks database existence,
 *   * creates tables, and provides ready-to-use connections.
 */
object DatabaseUtil {
  private val prop: Properties = new Properties()

  /** Loads database configuration from properties file and initializes the
   *   * driver. Executed only once.
   *
   */
  private def loadProperties(): Unit = {
    if (prop.isEmpty) {
      Using.resource(new FileInputStream("11-11-2025/DB.properties")) { file =>
        prop.load(file)
        Class.forName(prop.getProperty("DB_DRIVER_CLASS"))
      }
    }
  }

  /** Establishes a connection to the MySQL server (without selecting DB). */

```

```

private def getServerConnection(): Connection = {
    loadProperties()
    DriverManager.getConnection(
        prop.getProperty("DB_URL"),
        prop.getProperty("DB_USERNAME"),
        prop.getProperty("DB_PASSWORD")
    )
}

/** Returns a connection to the target database. Creates the database and
 * required tables if they do not exist.
 */
def getConnection(): Connection = {
    loadProperties()
    val dbName = prop.getProperty("DB_DATABASE_NAME")
    if (!databaseExists(dbName)) {
        createDatabase(dbName)
    }

    val conn = DriverManager.getConnection(
        s"${prop.getProperty("DB_URL")}/${dbName}",
        prop.getProperty("DB_USERNAME"),
        prop.getProperty("DB_PASSWORD")
    )

    val tableNames = List("Vehicles", "TrafficSignals", "Violations")
    if (tableNames.exists(tableName => !tableExists(conn, tableName))) {
        createTables(conn)
    }
    conn
}

/** Checks whether the target database exists.
 *
 * @param dbName
 *   Name of the database to check.
 */
private def databaseExists(dbName: String): Boolean = {
    Using.resource(getServerConnection()) { conn =>
        val query = "SHOW DATABASES LIKE ?"
        Using.resource(conn.prepareStatement(query)) { pstmt =>
            pstmt.setString(1, dbName)

```

```

        Using.resource(pstmt.executeQuery()) { rs =>
            rs.next()
        }
    }
}

/** Creates the database if it does not exist. */
private def createDatabase(dbName: String): Unit = {
    Using.resource(getServerConnection()) { conn =>
        Using.resource(conn.createStatement()) { stmt =>
            stmt.executeUpdate(s"CREATE DATABASE $dbName")
            println(s"Database '$dbName' created.")
        }
    }
}

/** Checks whether a particular table exists in the database.
 *
 * @param conn
 *   Active DB connection
 * @param tableName
 *   Name of the table to verify
 */
private def tableExists(conn: Connection, tableName: String): Boolean = {
    val query = "SHOW TABLES LIKE ?"
    Using.resource(conn.prepareStatement(query)) { pstmt =>
        pstmt.setString(1, tableName)
        Using.resource(pstmt.executeQuery()) { rs =>
            rs.next()
        }
    }
}

/** Creates tables Vehicles, TrafficSignals, and Violations if missing.
 *
 * @param conn
 *   Active connection to the database.
 */
private def createTables(conn: Connection): Unit = {
    val createVehiclesTable =
        """

```

```

        |CREATE TABLE IF NOT EXISTS Vehicles (
        | vehicle_id INT PRIMARY KEY AUTO_INCREMENT,
        | license_plate VARCHAR(20),
        | vehicle_type VARCHAR(20),
        | owner_name VARCHAR(100)
        |)
        |"".stripMargin

val createTrafficSignalsTable =
    """
        |CREATE TABLE IF NOT EXISTS TrafficSignals (
        | signal_id INT PRIMARY KEY AUTO_INCREMENT,
        | location VARCHAR(100),
        | status VARCHAR(10)
        |)
        |"".stripMargin

val createViolationsTable =
    """
        |CREATE TABLE IF NOT EXISTS Violations (
        | violation_id INT PRIMARY KEY AUTO_INCREMENT,
        | vehicle_id INT,
        | signal_id INT,
        | violation_type VARCHAR(50),
        | timestamp DATETIME,
        | FOREIGN KEY (vehicle_id) REFERENCES Vehicles(vehicle_id),
        | FOREIGN KEY (signal_id) REFERENCES TrafficSignals(signal_id)
        |)
        |"".stripMargin

Using.resource(conn.createStatement()) { stmt =>
    stmt.executeUpdate(createVehiclesTable)
    stmt.executeUpdate(createTrafficSignalsTable)
    stmt.executeUpdate(createViolationsTable)
    println("Tables created or already exist.")
}
}
}

/** Data Access Object (DAO) containing CRUD operations for Vehicles, Traffic
 * Signals, and Violations.
 *
 */

```

```

    * Provides high-level functions used by the main menu UI.
    */
object TrafficDAO {

    /** Inserts a new vehicle record.
     *
     * @param vehicle
     *     Vehicle data to insert.
     * @return
     *     true if insert succeeded, false otherwise.
     */
    def addVehicle(vehicle: Vehicle): Boolean = {
        val sql =
            "INSERT INTO Vehicles (license_plate, vehicle_type, owner_name) VALUES (?, ?, ?)"
        try {
            Using.resource(DatabaseUtil.getConnection()) { conn =>
                Using.resource(conn.prepareStatement(sql)) { pstmt =>
                    pstmt.setString(1, vehicle.licensePlate)
                    pstmt.setString(2, vehicle.vehicleType)
                    pstmt.setString(3, vehicle.ownerName)
                    pstmt.executeUpdate() > 0
                }
            }
        } catch {
            case e: Exception =>
                println(s"Error adding vehicle: ${e.getMessage}")
                false
        }
    }

    /** Inserts a new traffic signal into the database.
     *
     * @param signal
     *     Traffic signal to insert.
     */
    def addTrafficSignal(signal: TrafficSignal): Boolean = {
        val sql = "INSERT INTO TrafficSignals (location, status) VALUES (?, ?)"
        try {
            Using.resource(DatabaseUtil.getConnection()) { conn =>
                Using.resource(conn.prepareStatement(sql)) { pstmt =>
                    pstmt.setString(1, signal.location)
                    pstmt.setString(2, signal.status)
                }
            }
        }
    }
}

```

```

        pstmt.executeUpdate() > 0
    }
}
} catch {
    case _: Exception =>
        false
}
}

/** Records a traffic violation event. Ensures the referenced vehicle exists
 * before inserting.
 *
 * @param violation
 *     Violation details.
 * @return
 *     true if successful, false if vehicle does not exist or error.
 */
def recordViolation(violation: Violation): Boolean = {
    val checkVehicleSql = "SELECT vehicle_id FROM Vehicles WHERE vehicle_id = ?"
    val insertViolationSql =
        "INSERT INTO Violations (vehicle_id, signal_id, violation_type, timestamp) VALUES
        (?, ?, ?, ?)"
    try {
        Using.resource(DatabaseUtil.getConnection()) { conn =>
            Using.resource(conn.prepareStatement(checkVehicleSql)) { checkStmt =>
                checkStmt.setInt(1, violation.vehicleId)
                Using.resource(checkStmt.executeQuery()) { rs =>
                    if (rs.next()) {
                        Using.resource(conn.prepareStatement(insertViolationSql)) {
                            pstmt =>
                                pstmt.setInt(1, violation.vehicleId)
                                pstmt.setInt(2, violation.signalId)
                                pstmt.setString(3, violation.violationType)
                                pstmt.setTimestamp(4, Timestamp.valueOf(violation.timestamp))
                                pstmt.executeUpdate() > 0
                            }
                        } else {
                            false
                        }
                    }
                }
            }
        }
    }
}

```

```

    } catch {
        case _: Exception => false
    }
}

/** Updates the status of a traffic signal.
 *
 * @param signalId
 *     ID of the signal to update.
 * @param newStatus
 *     New status (green, yellow, red).
 */
def updateSignalStatus(signalId: Int, newStatus: String): Boolean = {
    val sql = "UPDATE TrafficSignals SET status = ? WHERE signal_id = ?"
    try {
        Using.resource(DatabaseUtil.getConnection()) { conn =>
            Using.resource(conn.prepareStatement(sql)) { pstmt =>
                pstmt.setString(1, newStatus)
                pstmt.setInt(2, signalId)
                pstmt.executeUpdate() > 0
            }
        }
    } catch {
        case _: Exception => false
    }
}

/** Retrieves all vehicles from the database. */
def viewVehicles(): List[Vehicle] = {
    val sql = "SELECT * FROM Vehicles"
    val vehicles = ListBuffer.empty[Vehicle]
    Using.resource(DatabaseUtil.getConnection()) { conn =>
        Using.resource(conn.createStatement()) { stmt =>
            Using.resource(stmt.executeQuery(sql)) { rs =>
                while (rs.next()) {
                    vehicles += Vehicle(
                        vehicleId = Some(rs.getInt("vehicle_id")),
                        licensePlate = rs.getString("license_plate"),
                        vehicleType = rs.getString("vehicle_type"),
                        ownerName = rs.getString("owner_name")
                    )
                }
            }
        }
    }
}

```

```

    }
    }
    }
    }
    vehicles.toList
}

/** Retrieves all traffic signals. */
def viewSignals(): List[TrafficSignal] = {
    val sql = "SELECT * FROM TrafficSignals"
    val signals = ListBuffer.empty[TrafficSignal]
    Using.resource(DatabaseUtil.getConnection()) { conn =>
        Using.resource(conn.createStatement()) { stmt =>
            Using.resource(stmt.executeQuery(sql)) { rs =>
                while (rs.next()) {
                    signals += TrafficSignal(
                        signalId = Some(rs.getInt("signal_id")),
                        location = rs.getString("location"),
                        status = rs.getString("status")
                    )
                }
            }
        }
    }
    signals.toList
}

/** Retrieves all recorded violations. */
def viewViolations(): List[Violation] = {
    val sql = "SELECT * FROM Violations"
    val violations = ListBuffer.empty[Violation]
    Using.resource(DatabaseUtil.getConnection()) { conn =>
        Using.resource(conn.createStatement()) { stmt =>
            Using.resource(stmt.executeQuery(sql)) { rs =>
                while (rs.next()) {
                    violations += Violation(
                        violationId = Some(rs.getInt("violation_id")),
                        vehicleId = rs.getInt("vehicle_id"),
                        signalId = rs.getInt("signal_id"),
                        violationType = rs.getString("violation_type"),
                        timestamp = rs.getTimestamp("timestamp").toLocalDateTime
                    )
                }
            }
        }
    }
    violations.toList
}

```

```

    )

    }

    }

    }

    }
    violations.toList
}

/** Deletes a vehicle by ID. */
def deleteVehicle(vehicleId: Int): Boolean = {
    val sql = "DELETE FROM Vehicles WHERE vehicle_id = ?"
    try {
        Using.resource(DatabaseUtil.getConnection()) { conn =>
            Using.resource(conn.prepareStatement(sql)) { pstmt =>
                pstmt.setInt(1, vehicleId)
                pstmt.executeUpdate() > 0
            }
        }
    } catch {
        case _: Exception => false
    }
}

/** Deletes a violation by ID. */
def deleteViolation(violationId: Int): Boolean = {
    val sql = "DELETE FROM Violations WHERE violation_id = ?"
    try {
        Using.resource(DatabaseUtil.getConnection()) { conn =>
            Using.resource(conn.prepareStatement(sql)) { pstmt =>
                pstmt.setInt(1, violationId)
                pstmt.executeUpdate() > 0
            }
        }
    } catch {
        case _: Exception => false
    }
}

/** Deletes a traffic signal by ID. */
def deleteTrafficSignal(signalId: Int): Boolean = {
    val sql = "DELETE FROM TrafficSignals WHERE signal_id = ?"

```

```

    try {
        Using.resource(DatabaseUtil.getConnection()) { conn =>
            Using.resource(conn.prepareStatement(sql)) { pstmt =>
                pstmt.setInt(1, signalId)
                pstmt.executeUpdate() > 0
            }
        }
    } catch {
        case _: Exception => false
    }
}

}

/** Console-based user interface loop for the Smart Traffic Management System.
 *
 * This interactive loop presents a menu of operations to the user and
 * dispatches requests to the [[TrafficDAO]] for CRUD and query operations.
 *
 * Menu options:
 * 1. Add Vehicle - Prompts for license plate, type and owner; inserts a
 * vehicle. 2. Delete Vehicle - Prompts for vehicle ID and deletes the
 * corresponding record. 3. Add Traffic Signal - Prompts for location and
 * status; inserts a traffic signal. 4. Delete Traffic Signal - Prompts
 * for signal ID and deletes the corresponding record. 5. Record Violation
 * \- Prompts for vehicle ID, signal ID and violation type; records a
 * violation. 6. Delete Violation - Prompts for violation ID and deletes
 * the corresponding record. 7. Update Signal Status - Prompts for signal
 * ID and new status; updates the signal record. 8. View Vehicles -
 * Displays a formatted list of all vehicles. 9. View Traffic Signals -
 * Displays a formatted list of all traffic signals.
 * 10. View Violations - Displays a formatted list of all recorded violations.
 * 11. Exit - Exits the interactive loop and terminates the program.
 */
object SmartTrafficManagementSystem extends App {
    var running = true

    def readInt(prompt: String): Option[Int] = {
        println(prompt)
        Try(scala.io.StdIn.readLine().trim.toInt) match {
            case Success(value) => Some(value)
            case Failure(_) =>

```

```

        println("Invalid number, please try again.")
        None
    }
}

def readNonEmptyString(prompt: String): Option[String] = {
    println(prompt)
    val input = scala.io.StdIn.readLine().trim
    if (input.nonEmpty) Some(input)
    else {
        println("Input cannot be empty.")
        None
    }
}

def validateStatus(status: String): Boolean = {
    Set("green", "yellow", "red").contains(status.toLowerCase)
}

def addVehicle(): Unit = {
    for {
        plate <- readNonEmptyString("License Plate:")
        vtype <- readNonEmptyString("Vehicle Type (car, bike, truck):")
        owner <- readNonEmptyString("Owner Name:")
    } {
        val added = TrafficDAO.addVehicle(
            Vehicle(licensePlate = plate, vehicleType = vtype, ownerName = owner)
        )
        if (added) println("Vehicle added successfully.")
        else println("Failed to add vehicle.")
    }
}

def deleteVehicle(): Unit = {
    readInt("Vehicle ID to delete:").foreach { vid =>
        val deleted = TrafficDAO.deleteVehicle(vid)
        if (deleted) println(s"Vehicle ID $vid deleted successfully.")
        else println(s"Vehicle ID $vid not found.")
    }
}

def addTrafficSignal(): Unit = {

```

```

for {
  location <- readNonEmptyString("Signal Location:")
  status <- {
    val s = scala.io.StdIn
      .readLine("Status (green, yellow, red):")
      .trim
      .toLowerCase
    if (validateStatus(s)) Some(s)
    else {
      println("Invalid status. Must be one of green, yellow, red.")
      None
    }
  }
} {
  val added = TrafficDAO.addTrafficSignal(
    TrafficSignal(location = location, status = status)
  )
  if (added) println("Traffic signal added successfully.")
  else println("Failed to add traffic signal.")
}

def deleteTrafficSignal(): Unit = {
  readInt("Traffic Signal ID to delete:").foreach { sid =>
    val deleted = TrafficDAO.deleteTrafficSignal(sid)
    if (deleted) println(s"Traffic Signal ID $sid deleted successfully.")
    else println(s"Traffic Signal ID $sid not found.")
  }
}

def recordViolation(): Unit = {
  for {
    vid <- readInt("Vehicle ID:")
    sid <- readInt("Signal ID:")
    vtype <- readNonEmptyString("Violation Type (speeding, signal jump):")
  } {
    val added = TrafficDAO.recordViolation(
      Violation(vehicleId = vid, signalId = sid, violationType = vtype)
    )
    if (added) println("Violation recorded successfully.")
    else
      println(s"Failed to record violation; vehicle ID $vid may not exist.")
  }
}

```

```

    }
}

def deleteViolation(): Unit = {
    readInt("Violation ID to delete:").foreach { violationId =>
        val deleted = TrafficDAO.deleteViolation(violationId)
        if (deleted) println(s"Violation ID $violationId deleted successfully.")
        else println(s"Violation ID $violationId not found.")
    }
}

def updateSignalStatus(): Unit = {
    for {
        sid <- readInt("Signal ID:")
        newStatus <- {
            val s = scala.io.StdIn
                .readLine("New Status (green, yellow, red):")
                .trim
                .toLowerCase
            if (validateStatus(s)) Some(s)
            else {
                println("Invalid status. Must be one of green, yellow, red.")
                None
            }
        }
    } {
        val updated = TrafficDAO.updateSignalStatus(sid, newStatus)
        if (updated) println("Signal status updated successfully.")
        else println("Signal ID not found or update failed.")
    }
}

def viewVehicles(): Unit = {
    val vehicles = TrafficDAO.viewVehicles()
    println(
        f"${"vehicle_id"}%-10s | ${"license_plate"}%-15s | ${"vehicle_type"}%-12s |
        ${"owner_name"}%-15s"
    )
    println("-" * 60)
    if (vehicles.nonEmpty) vehicles.foreach(println) else println("No Data.")
}

```

```

def viewSignals(): Unit = {
    val signals = TrafficDAO.viewSignals()
    println(f"${"signal_id"}%-10s | ${"location"}%-20s | ${"status"}%-10s")
    println("-" * 50)
    if (signals.nonEmpty) signals.foreach(println) else println("No Data.")
}

def viewViolations(): Unit = {
    val violations = TrafficDAO.viewViolations()
    println(
        f"${"violation_id"}%-10s | ${"vehicle_id"}%-10s | ${"signal_id"}%-10s |
        ${"violation_type"}%-15s | ${"timestamp"}%-20s"
    )
    println("-" * 90)
    if (violations.nonEmpty) violations.foreach(println)
    else println("No Data.")
}

while (running) {
    println(
        """
        |Smart Traffic Management System
        |1. Add Vehicle
        |2. Delete Vehicle
        |3. Add Traffic Signal
        |4. Delete Traffic Signal
        |5. Record Violation
        |6. Delete Violation
        |7. Update Signal Status
        |8. View Vehicles
        |9. View Traffic Signals
        |10. View Violations
        |11. Exit
        |Enter your choice:
        |""".stripMargin
    )

    scala.io.StdIn.readLine().trim.toIntOption match {
        case Some(choice) =>
            choice match {
                case 1 => addVehicle()
                case 2 => deleteVehicle()
            }
    }
}

```

```
    case 3 => addTrafficSignal()
    case 4 => deleteTrafficSignal()
    case 5 => recordViolation()
    case 6 => deleteViolation()
    case 7 => updateSignalStatus()
    case 8 => viewVehicles()
    case 9 => viewSignals()
    case 10 => viewViolations()
    case 11 =>
        println("Exiting...")
        running = false
    case _ => println("Invalid input. Try again.")
}
case None =>
    println("Please enter a valid number.")
}
}
```

Output:

```
PROBLEMS 52 OUTPUT DEBUG CONSOLE TERMINAL PORTS
rac@PTPMRT07 ScalaTraining % scala -classpath mysql-connector-j-8.4.0.jar 11-11-2025/SmartTrafficManagementSystem.scala
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
Compiling project (Scala 3.7.3, JVM (25))
Compiled project (Scala 3.7.3, JVM (25))

Smart Traffic Management System
1. Add Vehicle
2. Delete Vehicle
3. Add Traffic Signal
4. Delete Traffic Signal
5. Record Violation
6. Delete Violation
7. Update Signal Status
8. View Vehicles
9. View Traffic Signals
10. View Violations
11. Exit
Enter your choice:

1
License Plate:
AP12AB1234
Vehicle Type (car, bike, truck):
bike
Owner Name:
sanjeev
Vehicle added successfully.

Smart Traffic Management System
1. Add Vehicle
2. Delete Vehicle
3. Add Traffic Signal
4. Delete Traffic Signal
5. Record Violation
6. Delete Violation
7. Update Signal Status
8. View Vehicles
9. View Traffic Signals
10. View Violations
11. Exit
Enter your choice:

8
vehicle_id | license_plate | vehicle_type | owner_name
-----
1          | abc          | car          | Sanjeev
2          | abcd         | car          | kumar
3          | abcdf        | bike         | san
4          | AP09De123    | truck        | SanjeevKumarV
5          | AP12AB1234   | bike         | sanjeev

Smart Traffic Management System
```

nt Open Website Generate Commit Message BLACKBOX Autocomplete: DISABLED scala-cli ScalaTraining\_1f7954b597 ✓ BLACKBOX

```
PROBLEMS 52 OUTPUT DEBUG CONSOLE TERMINAL PORTS
rac@PTPMRT07 ScalaTraining % scala -classpath mysql-connector-j-8.4.0.jar 11-11-2025/SmartTrafficManagementSystem.scala
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
Compiling project (Scala 3.7.3, JVM (25))
Compiled project (Scala 3.7.3, JVM (25))

Smart Traffic Management System
1. Add Vehicle
2. Delete Vehicle
3. Add Traffic Signal
4. Delete Traffic Signal
5. Record Violation
6. Delete Violation
7. Update Signal Status
8. View Vehicles
9. View Traffic Signals
10. View Violations
11. Exit
Enter your choice:

1
License Plate:
AP12AB1234
Vehicle Type (car, bike, truck):
bike
Owner Name:
sanjeev
Vehicle added successfully.

Smart Traffic Management System
1. Add Vehicle
2. Delete Vehicle
3. Add Traffic Signal
4. Delete Traffic Signal
5. Record Violation
6. Delete Violation
7. Update Signal Status
8. View Vehicles
9. View Traffic Signals
10. View Violations
11. Exit
Enter your choice:

8
vehicle_id | license_plate | vehicle_type | owner_name
-----
1          | abc          | car          | Sanjeev
2          | abcd         | car          | kumar
3          | abcdf        | bike         | san
4          | AP09De123   | truck       | SanjeevKumarV
5          | AP12AB1234  | bike         | sanjeev

Smart Traffic Management System
nt Open Website Generate Commit Message  BLACKBOX Autocomplete: DISABLED scala-cli ScalaTraining_1f7954b597 BLACKBOX
```

## Q7. File Download Progress Simulator

Code:

```
/**
 * Represents a simulated file download task.
 *
 * Each task runs in its own thread and prints download progress in
 * 10% increments. The download speed determines how long the thread
 * sleeps between progress updates.
 *
 * @param fileName      Name of the file being downloaded.
 * @param downloadSpeed Delay (in milliseconds) per 10% progress chunk.
 */
case class DownloadTask(fileName: String, downloadSpeed: Int) extends Runnable {
```

```

/**
 * Executes the simulated download.
 *
 * The method:
 * - Iterates from 10% to 100% in steps of 10%
 * - Sleeps for `downloadSpeed` ms between each update
 * - Prints progress and a final completion message
 */
override def run(): Unit =
  for percent <- 10 to 100 by 10 do
    Thread.sleep(downloadSpeed)
    println(s"$fileName: $percent% downloaded")
    println(s"$fileName download completed!\n")
}

/**
 * Demonstrates concurrent file downloads using multiple threads.
 *
 * Features:
 * - Creates several `DownloadTask` instances
 * - Runs each task in its own thread
 * - Assigns different thread priorities
 * - Starts threads concurrently
 * - Waits for all downloads to finish using `join()`
 */
object FileDownloadProgressSimulator:

  /**
   * Entry point of the simulation.
   *
   * @param args Command-line arguments (unused)
   */
  def main(args: Array[String]): Unit =
    println("Starting file downloads...\n")

    // Create multiple download threads
    val file1 = new Thread(new DownloadTask("Movie.mp4", 300))
    val file2 = new Thread(new DownloadTask("Music.mp3", 200))
    val file3 = new Thread(new DownloadTask("Document.pdf", 500))
    val file4 = new Thread(new DownloadTask("GameInstaller.exe", 150))

    // Setting priority for threads

```

```
file1.setPriority(Thread.NORM_PRIORITY)
file2.setPriority(Thread.NORM_PRIORITY)
file3.setPriority(Thread.MAX_PRIORITY)
file4.setPriority(Thread.MIN_PRIORITY)

// Start all threads (run concurrently)
file1.start()
file2.start()
file3.start()
file4.start()

// Wait for all threads to complete before exiting
file1.join()
file2.join()
file3.join()
file4.join()

println("All downloads completed!")
```

Output:

```
rac@PTPMRT07 ScalaTraining % scala 11-11-2025/FileDownloadProgressSimulator.scala
WARNING: Please consider reporting this to the maintainers of class scala.runtime.LazyVals$
WARNING: sun.misc.Unsafe:objectFieldOffset will be removed in a future release
Compiling project (Scala 3.7.3, JVM (25))
Compiled project (Scala 3.7.3, JVM (25))
Starting file downloads...
```

```
GameInstaller.exe: 10% downloaded
Music.mp3: 10% downloaded
Movie.mp4: 10% downloaded
GameInstaller.exe: 20% downloaded
Music.mp3: 20% downloaded
GameInstaller.exe: 30% downloaded
Document.pdf: 10% downloaded
Movie.mp4: 20% downloaded
Music.mp3: 30% downloaded
GameInstaller.exe: 40% downloaded
GameInstaller.exe: 50% downloaded
Music.mp3: 40% downloaded
Movie.mp4: 30% downloaded
GameInstaller.exe: 60% downloaded
Document.pdf: 20% downloaded
Music.mp3: 50% downloaded
GameInstaller.exe: 70% downloaded
Movie.mp4: 40% downloaded
Music.mp3: 60% downloaded
GameInstaller.exe: 80% downloaded
GameInstaller.exe: 90% downloaded
Music.mp3: 70% downloaded
Document.pdf: 30% downloaded
Movie.mp4: 50% downloaded
GameInstaller.exe: 100% downloaded
GameInstaller.exe download completed!
```

```
Music.mp3: 80% downloaded
Movie.mp4: 60% downloaded
Music.mp3: 90% downloaded
Document.pdf: 40% downloaded
Music.mp3: 100% downloaded
Music.mp3 download completed!
```

```
Movie.mp4: 70% downloaded
Movie.mp4: 80% downloaded
Document.pdf: 50% downloaded
Movie.mp4: 90% downloaded
Document.pdf: 60% downloaded
Movie.mp4: 100% downloaded
Movie.mp4 download completed!
```

```
Document.pdf: 70% downloaded
Document.pdf: 80% downloaded
Document.pdf: 90% downloaded
Document.pdf: 100% downloaded
Document.pdf download completed!
```

```
All downloads completed!
```

```
rac@PTPMRT07 ScalaTraining %
```

nt Open Website Generate Commit Message

BLACKBOX Autocomplete: DISABLED

scala-cli

ScalaTraining\_1f7954b597 ✓

BLACKBOX