TELECOMMUNIVATION AND INFORMATION ENGINEERING

GROUP 4

MINI SIMULATION DEMO SOURCE CODE

```
import math

class Location:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance_to(self, other):
        return math.sqrt((self.x - other.x)**2 + (self.y - other.y)**2)

    def __str__(self):
        return f"({self.x}, {self.y})"

class BaseStation:
    def __init__(self, location, capacity=10, coverage_radius=100):
        self.location = location
        self.capacity = capacity
        self.coverage_radius = coverage_radius
        self.connected_devices = []

    def can_connect(self, device):
        dist = self.location.distance_to(device.location)
        if dist > self.coverage_radius:
            return False, "Out of range"
        if len(self.connected_devices) >= self.capacity:
            return False, "Capacity full"
        return True, ""

    def connect(self, device):
        can, reason = self.can_connect(device)
        if can:
            if device.connected_to:
                device.connected_to.disconnect(device)
            self.connected_devices.append(device)
            device.connected_to = self
            return True, "Connected successfully"
        return False, reason

    def disconnect(self, device):
        if device in self.connected_devices:
            self.connected_devices.remove(device)
            device.connected_to = None
            return True
        return False

    def get_signal_strength(self, device):
        dist = self.location.distance_to(device.location)
        if dist > self.coverage_radius:
            return 0
        load_factor = len(self.connected_devices) / self.capacity
        signal = 100 * (1 - dist / self.coverage_radius) * (1 - 0.5 * load_factor)
        return max(0, signal)
```

```
    def __str__(self):
        return f"BaseStation at {self.location}, connected: {len(self.connected_devices)}/{self.capacity}"

class Device:
    def __init__(self, device_id, device_type, location, battery_level=100):
        self.device_id = device_id
        self.device_type = device_type
        self.location = location
        self.battery_level = battery_level
        self.connected_to = None

    def _get_encapsulated_attrs(self):
        return {
            'battery_level': self.battery_level,
            'device_id': self.device_id,
            'signal_strength': self.get_signal_strength()
        }

    def connect(self):
        print(f"{self} attempting to connect...")
        for bs in base_stations:
            success, msg = bs.connect(self)
            if success:
                print(f"{self} {msg} to {bs}")
                return bs
        print(f"{self} failed to connect: no available base station")
        return None

    def disconnect(self):
        if self.connected_to:
            self.connected_to.disconnect(self)
            print(f"{self} disconnected from {self.connected_to}")
            self.connected_to = None

    def move(self, new_x, new_y):
        old_loc = self.location
        self.location = Location(new_x, new_y)
        print(f"{self} moved from {old_loc} to {self.location}")
        if self.connected_to:
            signal = self.get_signal_strength()
            if signal == 0:
                print("Signal lost due to move, triggering handover/disconnect")
                self.disconnect()
                self.connect()  # attempt handover

    def send_data(self):
        if not self.connected_to:
            print(f"{self} cannot send data: not connected")
            return
        print(f"{self} sending data via {self.connected_to}")
        self._perform_send()

    def _perform_send(self):
        print("    Generic data transfer")
```

```
    def get_signal_strength(self):
        if self.connected_to:
            return self.connected_to.get_signal_strength(self)
        return 0

    def __str__(self):
        return f"{self.device_type} [{self.device_id}] at {self.location}"

class SmartPhone(Device):
    def __init__(self, device_id, location, battery_level=100):
        super().__init__(device_id, "SmartPhone", location, battery_level)

    def _perform_send(self):
        print("    SmartPhone: High-speed data transfer / call simulation")

class IoTDevice(Device):
    def __init__(self, device_id, location, battery_level=100):
        super().__init__(device_id, "IoTDevice", location, battery_level)

    def _perform_send(self):
        print("    IoTDevice: Low-power sensor data upload")

class Drone(Device):
    def __init__(self, device_id, location, battery_level=100):
        super().__init__(device_id, "Drone", location, battery_level)

    def _perform_send(self):
        print("    Drone: Streaming video data")

# Global list of base stations (in a full app, this would be managed centrally)
base_stations = []
# ---------------- DEMO / MAIN PROGRAM ----------------

if __name__ == "__main__":
    print("=== Mini 5G Network Simulator Demo ===\n")

    # Create base stations
    bs1 = BaseStation(Location(0, 0), capacity=5)
    bs2 = BaseStation(Location(200, 0), capacity=3)

    base_stations.extend([bs1, bs2])

    print("Base stations created:")
    print(bs1)
    print(bs2)
    print()

    # Create devices
    phone = SmartPhone("PHONE-001", Location(20, 20))
    iot = IoTDevice("IOT-001", Location(30, 10))
    drone = Drone("DRONE-001", Location(50, 50))

    print("Devices registered:")
    print(phone)
```

```
    print(iot)
    print(drone)
    print()

    # Connect devices
    phone.connect()
    iot.connect()
    drone.connect()
    print()

    # Send data
    phone.send_data()
    iot.send_data()
    drone.send_data()
    print()

    # Move drone (handover demo)
    drone.move(180, 10)
    print(f"{drone} signal: {drone.get_signal_strength():.2f}%")
    print()

    # Encapsulation demo
    print("Encapsulation demo:")
    print(phone._get_encapsulated_attrs())

    print("\n=== Demo complete ===")
```