```
Week 4 Project Overview

Artifacts:
- Server Code with DAO using Sqllite using Spring Boot Data JPA with Hibernate
- React Client for interacting with the API
- Screen Shot of the React Client
- UML Diagram of the Application


================================================================================
FILE: src\main\java\com\cu5448\pcb\PcbApplication.java
================================================================================

package com.cu5448.pcb;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.scheduling.annotation.EnableAsync;

/**
 * Main Spring Boot Application with Configuration Properties Support
 *
 * <p>Demonstrates Dependency Injection design pattern implementation:
 * - @EnableConfigurationProperties enables property-driven configuration - REST API endpoints
 * available for running simulations on demand - All dependencies managed by Spring IoC container
 * - @EnableAsync enables asynchronous method execution for simulation processing
 *
 * <p>Server starts and waits for client API calls to run simulations and retrieve results.
 */
@SpringBootApplication
@EnableConfigurationProperties
@EnableAsync
public class PcbApplication {

    public static void main(String[] args) {
        SpringApplication.run(PcbApplication.class, args);
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\api\SimulationApiController.java
================================================================================

package com.cu5448.pcb.api;

import java.util.ArrayList;
import java.util.List;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import com.cu5448.pcb.controller.SimulationController;
import com.cu5448.pcb.dto.SimulationReportDto;
import com.cu5448.pcb.dto.SimulationReportMapper;
import com.cu5448.pcb.entity.SimulationResult;

import lombok.RequiredArgsConstructor;

/**
 * REST API Controller for PCB simulation operations. Implements the server-side requirement: run
 * simulations to gather failure results and persist them to SQLite database, then wait for client
 * API calls to retrieve the stored simulation results as JSON.
 *
 * <p>Provides separate endpoints for running simulations and retrieving persisted results from
 * database.
 */
@RestController
@RequestMapping("/api/simulation")
@CrossOrigin(origins = "http://localhost:3000")
@RequiredArgsConstructor
public class SimulationApiController {

    private final SimulationController simulationController;
    private final SimulationReportMapper reportMapper;

    /**
     * Start simulation for a specific PCB type asynchronously and store results in memory.
     *
     * @param pcbType the type of PCB to simulate (test, sensor, gateway)
     * @param quantity number of PCBs to process (default: 1000)
     * @return 201 Created status to indicate simulation started
     */
    @PostMapping("/run/{pcbType}")
    public ResponseEntity<String> runSimulation(
            @PathVariable String pcbType, @RequestParam(defaultValue = "1000") int quantity) {
```

```
        // Start simulation asynchronously using @Async method
        simulationController.runSimulationAsync(pcbType, quantity);

        return ResponseEntity.status(201)
                .body(
                        String.format(
                                "Simulation started asynchronously for PCB type: %s with quantity: %d",
                                pcbType, quantity));
    }

    /**
     * Start simulations for all three PCB types asynchronously and store results in memory.
     *
     * @param quantity number of PCBs to process for each type (default: 1000)
     * @return 201 Created status to indicate simulations started
     */
    @PostMapping("/run/all")
    public ResponseEntity<String> runAllSimulations(
            @RequestParam(defaultValue = "1000") int quantity) {

        // Start all simulations asynchronously using @Async method
        simulationController.runAllSimulationsAsync();

        return ResponseEntity.status(201)
                .body(
                        String.format(
                                "Simulations started asynchronously for all PCB types with quantity: %d",
                                quantity));
    }

    /**
     * Retrieve stored simulation results for a specific PCB type.
     *
     * @param pcbType the type of PCB to get results for
     * @return simulation report as JSON, or 404 if not found
     */
    @GetMapping("/results/{pcbType}")
    public ResponseEntity<SimulationReportDto> getSimulationResults(@PathVariable String pcbType) {
        SimulationResult result = simulationController.getSimulationResults(pcbType);

        if (result == null) {
            return ResponseEntity.notFound().build();
        }

        SimulationReportDto report = reportMapper.toDto(result);
        return ResponseEntity.ok(report);
    }

    /**
     * Retrieve all stored simulation results (latest for each PCB type).
     *
     * @return list of simulation reports for all stored results
     */
    @GetMapping("/results/all")
    public ResponseEntity<List<SimulationReportDto>> getAllSimulationResults() {
        List<SimulationResult> allResults = simulationController.getAllSimulationResults();

        if (allResults.isEmpty()) {
            return ResponseEntity.notFound().build();
        }

        List<SimulationReportDto> reports = new ArrayList<>();
        for (SimulationResult result : allResults) {
            SimulationReportDto report = reportMapper.toDto(result);
            reports.add(report);
        }

        return ResponseEntity.ok(reports);
    }

    /**
     * Get available PCB types.
     *
     * @return list of supported PCB types
     */
    @GetMapping("/types")
    public ResponseEntity<List<String>> getPcbTypes() {
        List<String> types = List.of("Test Board", "Sensor Board", "Gateway Board");
        return ResponseEntity.ok(types);
    }

    /**
     * Clear all stored simulation results.
     *
     * @return success response
     */
    @DeleteMapping("/results")
    public ResponseEntity<String> clearAllResults() {
        simulationController.clearAllResults();
```

```
        return ResponseEntity.ok("All simulation results cleared");
    }
}
```

```
================================================================================
FILE: src\main\java\com\cu5448\pcb\config\CorsConfig.java
================================================================================

package com.cu5448.pcb.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

/**
 * CORS configuration to allow React client (localhost:3000) to communicate with the Spring Boot
 * server (localhost:8080).
 */
@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
                .allowedOrigins("http://localhost:3000")
                .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
                .allowedHeaders("*")
                .allowCredentials(true);
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.addAllowedOrigin("http://localhost:3000");
        configuration.addAllowedMethod("*");
        configuration.addAllowedHeader("*");
        configuration.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/api/**", configuration);
        return source;
    }
}
```

```
================================================================================
FILE: src\main\java\com\cu5448\pcb\config\PCBProperties.java
================================================================================

package com.cu5448.pcb.config;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import com.cu5448.pcb.model.DefectRates;

import lombok.Data;

/**
 * PCB Configuration Properties that directly create DefectRates instances. This approach eliminates
 * nested property classes and provides direct access to DefectRates objects for each PCB type.
 */
@Data
@Component
@ConfigurationProperties(prefix = "pcb")
public class PCBProperties {

    private DefectRates testboard = new DefectRates();

    private DefectRates sensorboard = new DefectRates();

    private DefectRates gatewayboard = new DefectRates();
}
```

```
================================================================================
FILE: src\main\java\com\cu5448\pcb\config\PCBSimulationConfig.java
================================================================================

package com.cu5448.pcb.config;
```

```
import java.util.List;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.cu5448.pcb.factory.StationFactory;
import com.cu5448.pcb.station.*;

import lombok.RequiredArgsConstructor;

/**
 * Spring Configuration Class using Abstract Factory Pattern for PCB Assembly Line Stations. The
 * configuration delegates station creation to a StationFactory, maintaining the factory pattern
 * while leveraging Spring's dependency injection.
 */
@Configuration
@RequiredArgsConstructor
public class PCBSimulationConfig {

    private final StationFactory stationFactory;

    /**
     * Creates ordered list of stations for the assembly line using the abstract factory pattern.
     * This ensures consistent station creation and proper manufacturing process flow.
     */
    @Bean
    public List<Station> createAssemblyLineStations() {
        return List.of(
                stationFactory.createStation("ApplySolderPaste"),
                stationFactory.createStation("PlaceComponents"),
                stationFactory.createStation("ReflowSolder"),
                stationFactory.createStation("OpticalInspection"),
                stationFactory.createStation("HandSoldering"),
                stationFactory.createStation("Cleaning"),
                stationFactory.createStation("Depanelization"),
                stationFactory.createStation("Test"));
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\config\StationProperties.java
================================================================================

package com.cu5448.pcb.config;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import lombok.Data;

/** Station Configuration Properties using Lombok @Data generates all necessary boilerplate code */
@Data
@Component
@ConfigurationProperties(prefix = "station")
public class StationProperties {

    private double failureRate = 0.002;
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\controller\SimulationController.java
================================================================================

package com.cu5448.pcb.controller;

import java.util.List;
import java.util.concurrent.CompletableFuture;

import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Component;

import com.cu5448.pcb.entity.SimulationResult;
import com.cu5448.pcb.repository.SimulationResultRepository;
import com.cu5448.pcb.service.AssemblyLine;
import com.cu5448.pcb.service.StatisticsCollector;

import lombok.RequiredArgsConstructor;

/**
 * Main Simulation Controller using Spring Dependency Injection and Lombok @RequiredArgsConstructor
 * generates constructor for final fields. This controller orchestrates PCB simulations and manages
 * results storage in SQLite database via JPA for REST API access.
 *
 * <p>Supports the server-side requirement: run simulations to gather failure results and persist
 * them to database, then wait for client API calls to retrieve the stored simulation results.
```

```
 */
@Component
@RequiredArgsConstructor
public class SimulationController {

    private final AssemblyLine assemblyLine;
    private final SimulationResultRepository simulationResultRepository;

    /**
     * Run simulation for a specific PCB type asynchronously and persist results to database.
     *
     * @param pcbType the type of PCB to simulate
     * @param quantity number of PCBs to process
     * @return CompletableFuture with the simulation result entity
     */
    @Async
    public CompletableFuture<SimulationResult> runSimulationAsync(String pcbType, int quantity) {
        StatisticsCollector stats = assemblyLine.runSimulation(pcbType, quantity);
        SimulationResult result = createAndSaveSimulationResult(stats, pcbType, quantity);
        return CompletableFuture.completedFuture(result);
    }

    /**
     * Run simulation for a specific PCB type synchronously and persist results to database.
     *
     * @param pcbType the type of PCB to simulate
     * @param quantity number of PCBs to process
     * @return the simulation result entity
     */
    public SimulationResult runSimulation(String pcbType, int quantity) {
        StatisticsCollector stats = assemblyLine.runSimulation(pcbType, quantity);
        return createAndSaveSimulationResult(stats, pcbType, quantity);
    }

    /**
     * Run simulation for a specific PCB type with default quantity (1000).
     *
     * @param pcbType the type of PCB to simulate
     * @return the simulation result entity
     */
    public SimulationResult runSimulation(String pcbType) {
        return runSimulation(pcbType, 1000);
    }

    /**
     * Helper method to create and save SimulationResult from StatisticsCollector.
     *
     * @param stats the statistics collector with simulation data
     * @param pcbType the PCB type that was simulated
     * @param quantity the quantity of PCBs processed
     * @return the saved simulation result entity
     */
    private SimulationResult createAndSaveSimulationResult(
            StatisticsCollector stats, String pcbType, int quantity) {
        SimulationResult result =
                new SimulationResult(
                        pcbType,
                        stats.getPcbsSubmitted(),
                        stats.getCompletedPCBs(),
                        stats.getStationFailures(),
                        stats.getDefectFailures(),
                        stats.generateReport(pcbType),
                        quantity);
        return simulationResultRepository.save(result);
    }

    /**
     * Run simulations for all three PCB types asynchronously and persist results to database.
     *
     * @return CompletableFuture that completes when all simulations are done
     */
    @Async
    public CompletableFuture<List<SimulationResult>> runAllSimulationsAsync() {
        SimulationResult testResult = runSimulation("test");
        SimulationResult sensorResult = runSimulation("sensor");
        SimulationResult gatewayResult = runSimulation("gateway");
        return CompletableFuture.completedFuture(List.of(testResult, sensorResult, gatewayResult));
    }

    /**
     * Retrieve the most recent simulation result for a specific PCB type.
     *
     * @param pcbType the type of PCB to get results for
     * @return the most recent simulation result, or null if not found
     */
    public SimulationResult getSimulationResults(String pcbType) {
        return simulationResultRepository
                .findFirstByPcbTypeOrderByCreatedAtDesc(pcbType)
                .orElse(null);
```

```
    }

    /**
     * Retrieve the most recent simulation result for each PCB type.
     *
     * @return list of the latest simulation results for each PCB type
     */
    public List<SimulationResult> getAllSimulationResults() {
        return simulationResultRepository.findLatestResultForEachPcbType();
    }

    /** Clear all stored simulation results from database. */
    public void clearAllResults() {
        simulationResultRepository.deleteAll();
    }
}
```

```
================================================================================
FILE: src\main\java\com\cu5448\pcb\dto\SimulationReportDto.java
================================================================================

package com.cu5448.pcb.dto;

import java.util.Map;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * Data Transfer Object for simulation report data used in REST API communication between server and
 * client applications.
 */
@Data
@NoArgsConstructor
@AllArgsConstructor
public class SimulationReportDto {

    @JsonProperty("pcbType")
    private String pcbType;

    @JsonProperty("pcbsRun")
    private int pcbsRun;

    @JsonProperty("stationFailures")
    private Map<String, Integer> stationFailures;

    @JsonProperty("defectFailures")
    private Map<String, Integer> defectFailures;

    @JsonProperty("totalFailedPcbs")
    private int totalFailedPcbs;

    @JsonProperty("totalPcbsProduced")
    private int totalPcbsProduced;

    @JsonProperty("formattedReport")
    private String formattedReport;
}
```

```
================================================================================
FILE: src\main\java\com\cu5448\pcb\dto\SimulationReportMapper.java
================================================================================

package com.cu5448.pcb.dto;

import org.springframework.stereotype.Component;

import com.cu5448.pcb.entity.SimulationResult;
import com.cu5448.pcb.service.StatisticsCollector;

/**
 * Mapper class to convert SimulationResult entities and StatisticsCollector data to
 * SimulationReportDto for API communication.
 */
@Component
public class SimulationReportMapper {

    /**
     * Convert SimulationResult entity to DTO.
     *
     * @param result the simulation result entity
     * @return the DTO for API response
```

```
     */
    public SimulationReportDto toDto(SimulationResult result) {
        SimulationReportDto dto = new SimulationReportDto();

        dto.setPcbType(result.getPcbType());
        dto.setPcbsRun(result.getPcbsSubmitted());
        dto.setStationFailures(result.getStationFailures());
        dto.setDefectFailures(result.getDefectFailures());
        dto.setTotalFailedPcbs(result.getTotalFailedPcbs());
        dto.setTotalPcbsProduced(result.getPcbsCompleted());
        dto.setFormattedReport(result.getFormattedReport());

        return dto;
    }

    /**
     * Convert StatisticsCollector data to DTO (for backward compatibility).
     *
     * @param stats the statistics collector
     * @param pcbType the PCB type
     * @return the DTO for API response
     */
    public SimulationReportDto toDto(StatisticsCollector stats, String pcbType) {
        SimulationReportDto dto = new SimulationReportDto();

        dto.setPcbType(pcbType);
        dto.setPcbsRun(stats.getPcbsSubmitted());
        dto.setStationFailures(stats.getStationFailures());
        dto.setDefectFailures(stats.getDefectFailures());
        dto.setTotalFailedPcbs(stats.getPcbsSubmitted() - stats.getCompletedPCBs());
        dto.setTotalPcbsProduced(stats.getCompletedPCBs());
        dto.setFormattedReport(stats.generateReport(pcbType));

        return dto;
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\entity\MapToJsonConverter.java
================================================================================

package com.cu5448.pcb.entity;

import java.util.HashMap;
import java.util.Map;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;

import jakarta.persistence.AttributeConverter;
import jakarta.persistence.Converter;

/**
 * JPA AttributeConverter to store Map<String, Integer> as JSON string in database. This allows us
 * to store complex data structures in SQLite which doesn't have native JSON support.
 */
@Converter
public class MapToJsonConverter implements AttributeConverter<Map<String, Integer>, String> {

    private static final ObjectMapper objectMapper = new ObjectMapper();

    @Override
    public String convertToDatabaseColumn(Map<String, Integer> attribute) {
        if (attribute == null || attribute.isEmpty()) {
            return "{}";
        }
        try {
            return objectMapper.writeValueAsString(attribute);
        } catch (JsonProcessingException e) {
            throw new RuntimeException("Error converting Map to JSON", e);
        }
    }

    @Override
    public Map<String, Integer> convertToEntityAttribute(String dbData) {
        if (dbData == null || dbData.trim().isEmpty()) {
            return new HashMap<>();
        }
        try {
            return objectMapper.readValue(dbData, new TypeReference<Map<String, Integer>>() {});
        } catch (JsonProcessingException e) {
            throw new RuntimeException("Error converting JSON to Map", e);
        }
    }
}
```

```
================================================================================
FILE: src\main\java\com\cu5448\pcb\entity\SimulationResult.java
================================================================================

package com.cu5448.pcb.entity;

import java.time.LocalDateTime;
import java.util.Map;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * JPA Entity for persisting simulation results to SQLite database. Stores all statistical data from
 * StatisticsCollector along with metadata.
 */
@Entity
@Table(name = "simulation_results")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class SimulationResult {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "pcb_type", nullable = false, length = 50)
    private String pcbType;

    @Column(name = "pcbs_submitted", nullable = false)
    private int pcbsSubmitted;

    @Column(name = "pcbs_completed", nullable = false)
    private int pcbsCompleted;

    @Column(name = "total_failed_pcbs", nullable = false)
    private int totalFailedPcbs;

    @Column(name = "station_failures", columnDefinition = "TEXT")
    @Convert(converter = MapToJsonConverter.class)
    private Map<String, Integer> stationFailures;

    @Column(name = "defect_failures", columnDefinition = "TEXT")
    @Convert(converter = MapToJsonConverter.class)
    private Map<String, Integer> defectFailures;

    @Column(name = "formatted_report", columnDefinition = "TEXT")
    private String formattedReport;

    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt;

    @Column(name = "simulation_quantity", nullable = false)
    private int simulationQuantity;

    /** Constructor for creating SimulationResult from StatisticsCollector data. */
    public SimulationResult(
            String pcbType,
            int pcbsSubmitted,
            int pcbsCompleted,
            Map<String, Integer> stationFailures,
            Map<String, Integer> defectFailures,
            String formattedReport,
            int simulationQuantity) {
        this.pcbType = pcbType;
        this.pcbsSubmitted = pcbsSubmitted;
        this.pcbsCompleted = pcbsCompleted;
        this.totalFailedPcbs = pcbsSubmitted - pcbsCompleted;
        this.stationFailures = stationFailures;
        this.defectFailures = defectFailures;
        this.formattedReport = formattedReport;
        this.simulationQuantity = simulationQuantity;
        this.createdAt = LocalDateTime.now();
    }

    @PrePersist
    protected void onCreate() {
        if (createdAt == null) {
            createdAt = LocalDateTime.now();
        }
        // Calculate total failed PCBs if not set
        if (totalFailedPcbs == 0 && pcbsSubmitted > 0) {
            totalFailedPcbs = pcbsSubmitted - pcbsCompleted;
        }
```

```
    }
}
```

```
================================================================================
FILE: src\main\java\com\cu5448\pcb\factory\PCBFactory.java
================================================================================

package com.cu5448.pcb.factory;

import org.springframework.stereotype.Component;

import com.cu5448.pcb.config.PCBProperties;
import com.cu5448.pcb.model.GatewayBoard;
import com.cu5448.pcb.model.PCB;
import com.cu5448.pcb.model.SensorBoard;
import com.cu5448.pcb.model.TestBoard;

import lombok.RequiredArgsConstructor;

/**
 * Factory Pattern Implementation using Spring Dependency Injection and
 * Lombok @RequiredArgsConstructor generates constructor for final fields This factory creates PCB
 * instances with configuration-driven defect rates.
 */
@Component
@RequiredArgsConstructor
public class PCBFactory {

    private final PCBProperties pcbProperties;

    public PCB createPCB(String type) {
        return switch (type.toLowerCase()) {
            case "testboard", "test", "test board" -> new TestBoard(pcbProperties.getTestboard());
            case "sensorboard", "sensor", "sensor board" ->
                    new SensorBoard(pcbProperties.getSensorboard());
            case "gatewayboard", "gateway", "gateway board" ->
                    new GatewayBoard(pcbProperties.getGatewayboard());
            default -> throw new IllegalArgumentException("Unknown PCB type: " + type);
        };
    }
}
```

```
================================================================================
FILE: src\main\java\com\cu5448\pcb\factory\StationFactory.java
================================================================================

package com.cu5448.pcb.factory;

import java.util.Map;
import java.util.function.Function;

import org.springframework.stereotype.Component;

import com.cu5448.pcb.config.StationProperties;
import com.cu5448.pcb.station.*;

import lombok.RequiredArgsConstructor;

/**
 * Abstract Factory for creating PCB manufacturing stations using Spring Dependency Injection. This
 * factory uses a registry pattern to eliminate the need for individual creation methods for each
 * station type, making it more extensible and following the Abstract Factory pattern.
 */
@Component
@RequiredArgsConstructor
public class StationFactory {

    private final StationProperties stationProperties;

    // Registry of station constructors using method references
    private final Map<String, Function<Double, Station>> stationRegistry =
            Map.of(
                    "ApplySolderPaste", ApplySolderPasteStation::new,
                    "PlaceComponents", PlaceComponentsStation::new,
                    "ReflowSolder", ReflowSolderStation::new,
                    "OpticalInspection", OpticalInspectionStation::new,
                    "HandSoldering", HandSolderingStation::new,
                    "Cleaning", CleaningStation::new,
                    "Depanelization", DepanelizationStation::new,
                    "Test", TestStation::new);

    /**
     * Creates a station by type name using the Abstract Factory pattern. This method uses a
     * registry of constructor method references to eliminate the need for individual creation
     * methods.
```

```
     *
     * @param stationType the type of station to create (e.g., "ApplySolderPaste", "Test")
     * @return Station instance of the specified type
     * @throws IllegalArgumentException if station type is unknown
     */
    public Station createStation(String stationType) {
        Function<Double, Station> constructor = stationRegistry.get(stationType);
        if (constructor == null) {
            throw new IllegalArgumentException("Unknown station type: " + stationType);
        }
        return constructor.apply(stationProperties.getFailureRate());
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\model\DefectRates.java
================================================================================

package com.cu5448.pcb.model;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * DefectRates encapsulates defect rates for different manufacturing stations. This class replaces
 * the Map<String, Double> approach with a type-safe, immutable object.
 *
 * <p>Only four stations can detect defects: PlaceComponents, OpticalInspection, HandSoldering, and
 * Test.
 */
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class DefectRates {

    private double placeComponentsDefectRate;
    private double opticalInspectionDefectRate;
    private double handSolderingDefectRate;
    private double testDefectRate;

    /**
     * Gets the defect rate for a specific station type.
     *
     * @param stationType the station type name
     * @return the defect rate for the station, or 0.0 if the station doesn't detect defects
     */
    public double getDefectRate(String stationType) {
        return switch (stationType) {
            case "PlaceComponents" -> placeComponentsDefectRate;
            case "OpticalInspection" -> opticalInspectionDefectRate;
            case "HandSoldering" -> handSolderingDefectRate;
            case "Test" -> testDefectRate;
            default -> 0.0; // Stations that don't detect defects
        };
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\model\GatewayBoard.java
================================================================================

package com.cu5448.pcb.model;

import lombok.EqualsAndHashCode;

/** Gateway Board PCB Implementation using Lombok */
@EqualsAndHashCode(callSuper = true)
public class GatewayBoard extends PCB {

    private final DefectRates defectRates;

    public GatewayBoard(DefectRates defectRates) {
        super("GatewayBoard");
        this.defectRates = defectRates;
    }

    @Override
    public double getDefectRate(String stationType) {
        return defectRates.getDefectRate(stationType);
    }

    @Override
```

```
    public DefectRates getDefectRates() {
        return defectRates;
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\model\PCB.java
================================================================================

package com.cu5448.pcb.model;

import java.util.UUID;

import lombok.Getter;
import lombok.ToString;

/**
 * Abstract PCB Model using Lombok @Getter generates getters for all fields @ToString generates
 * toString method
 */
@Getter
@ToString
public abstract class PCB {

    private final String id;

    private final String type;

    private boolean failed = false;

    private String failureReason = null;

    public PCB(String type) {
        this.id = UUID.randomUUID().toString();
        this.type = type;
    }

    public void setFailed(String reason) {
        this.failed = true;
        this.failureReason = reason;
    }

    public abstract double getDefectRate(String stationType);

    public abstract DefectRates getDefectRates();
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\model\SensorBoard.java
================================================================================

package com.cu5448.pcb.model;

import lombok.EqualsAndHashCode;

/** Sensor Board PCB Implementation using Lombok */
@EqualsAndHashCode(callSuper = true)
public class SensorBoard extends PCB {

    private final DefectRates defectRates;

    public SensorBoard(DefectRates defectRates) {
        super("SensorBoard");
        this.defectRates = defectRates;
    }

    @Override
    public double getDefectRate(String stationType) {
        return defectRates.getDefectRate(stationType);
    }

    @Override
    public DefectRates getDefectRates() {
        return defectRates;
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\model\TestBoard.java
================================================================================

package com.cu5448.pcb.model;
```

```
import lombok.EqualsAndHashCode;

/** Test Board PCB Implementation using Lombok */
@EqualsAndHashCode(callSuper = true)
public class TestBoard extends PCB {

    private final DefectRates defectRates;

    public TestBoard(DefectRates defectRates) {
        super("TestBoard");
        this.defectRates = defectRates;
    }

    @Override
    public double getDefectRate(String stationType) {
        return defectRates.getDefectRate(stationType);
    }

    @Override
    public DefectRates getDefectRates() {
        return defectRates;
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\repository\SimulationResultRepository.java
================================================================================

package com.cu5448.pcb.repository;

import java.util.List;
import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import com.cu5448.pcb.entity.SimulationResult;

/**
 * Spring Data JPA repository for SimulationResult entities. Provides CRUD operations and custom
 * queries for simulation result data.
 */
@Repository
public interface SimulationResultRepository extends JpaRepository<SimulationResult, Long> {

    /**
     * Find the most recent simulation result for a specific PCB type.
     *
     * @param pcbType the PCB type to search for
     * @return the most recent SimulationResult for the given PCB type, if any
     */
    Optional<SimulationResult> findFirstByPcbTypeOrderByCreatedAtDesc(String pcbType);

    /**
     * Get the latest simulation result for each PCB type.
     *
     * @return list of the most recent SimulationResult for each PCB type
     */
    @Query(
            "SELECT sr FROM SimulationResult sr WHERE sr.createdAt = "
                    + "(SELECT MAX(sr2.createdAt) FROM SimulationResult sr2 WHERE sr2.pcbType ="
                    + " sr.pcbType)")
    List<SimulationResult> findLatestResultForEachPcbType();
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\service\AssemblyLine.java
================================================================================

package com.cu5448.pcb.service;

import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Service;

import com.cu5448.pcb.factory.PCBFactory;
import com.cu5448.pcb.model.PCB;
import com.cu5448.pcb.station.Station;

import lombok.RequiredArgsConstructor;

/**
 * Assembly Line Service using Spring Dependency Injection. Station beans are injected as an ordered
```

```
 * list, eliminating the need for manual station creation and initialization.
 */
@Service
@RequiredArgsConstructor
public class AssemblyLine {

    private final List<Station> stations;

    private final PCBFactory factory;

    private final ApplicationContext applicationContext;

    public void processPCB(PCB pcb, StatisticsCollector stats) {
        for (Station station : stations) {
            station.process(pcb, stats);
            if (pcb.isFailed()) {
                break;
            }
        }
    }

    public StatisticsCollector runSimulation(String pcbType, int quantity) {
        // Get a new prototype instance of StatisticsCollector for this simulation
        StatisticsCollector stats = applicationContext.getBean(StatisticsCollector.class);

        for (int i = 0; i < quantity; i++) {
            PCB pcb = factory.createPCB(pcbType);
            stats.recordSubmission();

            processPCB(pcb, stats);

            if (!pcb.isFailed()) {
                stats.recordCompletion();
            }
        }

        return stats;
    }

    public List<Station> getStations() {
        return List.copyOf(stations);
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\service\StatisticsCollector.java
================================================================================

package com.cu5448.pcb.service;

import java.util.HashMap;
import java.util.Map;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Service;

import lombok.Getter;

/**
 * Observer Pattern Implementation as Spring Service using Lombok @Getter generates getter methods
 * for all fields This service observes events from stations during PCB processing. Uses prototype
 * scope to create new instances for each simulation run.
 */
@Service
@Scope("prototype")
@Getter
public class StatisticsCollector {

    private int pcbsSubmitted;

    private final Map<String, Integer> defectFailures;

    private final Map<String, Integer> stationFailures;

    private int completedPCBs;

    public StatisticsCollector() {
        this.pcbsSubmitted = 0;
        this.defectFailures = new HashMap<>();
        this.stationFailures = new HashMap<>();
        this.completedPCBs = 0;
    }

    public void recordSubmission() {
        pcbsSubmitted++;
    }
```

```
    public void recordDefectFailure(String station) {
        defectFailures.merge(station, 1, Integer::sum);
    }

    public void recordStationFailure(String station) {
        stationFailures.merge(station, 1, Integer::sum);
    }

    public void recordCompletion() {
        completedPCBs++;
    }

    public String generateReport(String pcbType) {
        StringBuilder report = new StringBuilder();

        // Format according to project specification
        report.append(String.format("PCB type: %s\n", pcbType));
        report.append(String.format("PCBs run: %d\n", pcbsSubmitted));

        report.append("\nStation Failures\n");
        // Show all stations in assembly order
        String[] stationNames = {
            "Apply Solder Paste",
            "Place Components",
            "Reflow Solder",
            "Optical Inspection",
            "Hand Soldering/Assembly",
            "Cleaning",
            "Depanelization",
            "Test (ICT or Flying Probe)"
        };

        String[] stationKeys = {
            "ApplySolderPaste",
            "PlaceComponents",
            "ReflowSolder",
            "OpticalInspection",
            "HandSoldering",
            "Cleaning",
            "Depanelization",
            "Test"
        };

        for (int i = 0; i < stationNames.length; i++) {
            int failures = stationFailures.getOrDefault(stationKeys[i], 0);
            report.append(String.format("%s: %d\n", stationNames[i], failures));
        }

        report.append("\nPCB Defect Failures\n");
        // Only show defect-detecting stations
        String[] defectStationNames = {
            "Place Components",
            "Optical Inspection",
            "Hand Soldering/Assembly",
            "Test (ICT or Flying Probe)"
        };
        String[] defectStationKeys = {
            "PlaceComponents", "OpticalInspection", "HandSoldering", "Test"
        };

        for (int i = 0; i < defectStationNames.length; i++) {
            int failures = defectFailures.getOrDefault(defectStationKeys[i], 0);
            report.append(String.format("%s %d\n", defectStationNames[i], failures));
        }

        // Calculate total failures and successful PCBs
        int totalFailed = pcbsSubmitted - completedPCBs;

        report.append("\nFinal Results\n");
        report.append(String.format("Total failed PCBs: %d\n", totalFailed));
        report.append(String.format("Total PCBs produced: %d\n", completedPCBs));

        return report.toString();
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\station\ApplySolderPasteStation.java
================================================================================

package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

public class ApplySolderPasteStation extends Station {

    public ApplySolderPasteStation(double failureRate) {
```

```
        super("ApplySolderPaste", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        return true;
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\station\CleaningStation.java
================================================================================

package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

public class CleaningStation extends Station {

    public CleaningStation(double failureRate) {
        super("Cleaning", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        return true;
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\station\DepanelizationStation.java
================================================================================

package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

public class DepanelizationStation extends Station {

    public DepanelizationStation(double failureRate) {
        super("Depanelization", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        return true;
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\station\HandSolderingStation.java
================================================================================

package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

import lombok.EqualsAndHashCode;

/** Hand Soldering Station using Lombok */
@EqualsAndHashCode(callSuper = true)
public class HandSolderingStation extends Station {

    public HandSolderingStation(double failureRate) {
        super("HandSoldering", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        double defectRate = pcb.getDefectRate("HandSoldering");
        return random.nextDouble() >= defectRate;
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\station\OpticalInspectionStation.java
================================================================================

package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;
```

```
import lombok.EqualsAndHashCode;

/** Optical Inspection Station using Lombok */
@EqualsAndHashCode(callSuper = true)
public class OpticalInspectionStation extends Station {

    public OpticalInspectionStation(double failureRate) {
        super("OpticalInspection", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        double defectRate = pcb.getDefectRate("OpticalInspection");
        return random.nextDouble() >= defectRate;
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\station\PlaceComponentsStation.java
================================================================================

package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

import lombok.EqualsAndHashCode;

/**
 * Place Components Station using Lombok @EqualsAndHashCode(callSuper = true) includes parent class
 * fields in equals/hashCode
 */
@EqualsAndHashCode(callSuper = true)
public class PlaceComponentsStation extends Station {

    public PlaceComponentsStation(double failureRate) {
        super("PlaceComponents", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        double defectRate = pcb.getDefectRate("PlaceComponents");
        return random.nextDouble() >= defectRate;
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\station\ReflowSolderStation.java
================================================================================

package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

public class ReflowSolderStation extends Station {

    public ReflowSolderStation(double failureRate) {
        super("ReflowSolder", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        return true;
    }
}



================================================================================
FILE: src\main\java\com\cu5448\pcb\station\Station.java
================================================================================

package com.cu5448.pcb.station;

import java.util.Random;

import com.cu5448.pcb.model.PCB;
import com.cu5448.pcb.service.StatisticsCollector;

import lombok.Getter;

/**
 * Abstract Station class that can be used as a Spring bean. StatisticsCollector is injected per
 * simulation run rather than at construction time.
 */
```

```
@Getter
public abstract class Station {

    protected final String name;

    protected final double stationFailureRate;

    protected final Random random = new Random();

    public Station(String name, double failureRate) {
        this.name = name;
        this.stationFailureRate = failureRate;
    }

    public void process(PCB pcb, StatisticsCollector stats) {
        if (pcb.isFailed()) {
            return;
        }

        if (checkStationFailure()) {
            stats.recordStationFailure(name);
            pcb.setFailed("Station failure at " + name);
            return;
        }

        boolean operationSuccessful = performOperation(pcb);
        if (!operationSuccessful) {
            stats.recordDefectFailure(name);
            pcb.setFailed("Defect detected at " + name);
        }
    }

    protected boolean checkStationFailure() {
        return random.nextDouble() < stationFailureRate;
    }

    protected abstract boolean performOperation(PCB pcb);
}
```

```
================================================================================
FILE: src\main\java\com\cu5448\pcb\station\TestStation.java
================================================================================

package com.cu5448.pcb.station;

import com.cu5448.pcb.model.PCB;

import lombok.EqualsAndHashCode;

/** Test Station using Lombok */
@EqualsAndHashCode(callSuper = true)
public class TestStation extends Station {

    public TestStation(double failureRate) {
        super("Test", failureRate);
    }

    @Override
    protected boolean performOperation(PCB pcb) {
        double defectRate = pcb.getDefectRate("Test");
        return random.nextDouble() >= defectRate;
    }
}
```

```
================================================================================
FILE: src\test\java\com\cu5448\pcb\PcbApplicationTests.java
================================================================================

package com.cu5448.pcb;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class PcbApplicationTests {

    @Test
    void contextLoads() {}
}
```

```
================================================================================
FILE: src\test\java\com\cu5448\pcb\config\SpringBeanConfigurationTest.java
================================================================================
```

```
package com.cu5448.pcb.config;

import static org.junit.jupiter.api.Assertions.*;

import java.util.List;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.TestConstructor;

import com.cu5448.pcb.service.AssemblyLine;
import com.cu5448.pcb.station.*;

import lombok.RequiredArgsConstructor;

/** Test class to verify Spring bean configuration using Abstract Factory Pattern */
@SpringBootTest
@TestConstructor(autowireMode = TestConstructor.AutowireMode.ALL)
@RequiredArgsConstructor
class SpringBeanConfigurationTest {

    private final AssemblyLine assemblyLine;

    private final List<Station> stations;

    @Test
    void testAssemblyLineInjection() {
        assertNotNull(assemblyLine, "AssemblyLine should be injected");

        List<Station> assemblyStations = assemblyLine.getStations();
        assertEquals(8, assemblyStations.size(), "Assembly line should have 8 stations");
    }

    @Test
    void testStationListOrder() {
        assertNotNull(stations, "Station list should be injected");
        assertEquals(8, stations.size(), "Should have 8 stations");

        // Verify the correct order of stations
        assertEquals("ApplySolderPaste", stations.get(0).getName());
        assertEquals("PlaceComponents", stations.get(1).getName());
        assertEquals("ReflowSolder", stations.get(2).getName());
        assertEquals("OpticalInspection", stations.get(3).getName());
        assertEquals("HandSoldering", stations.get(4).getName());
        assertEquals("Cleaning", stations.get(5).getName());
        assertEquals("Depanelization", stations.get(6).getName());
        assertEquals("Test", stations.get(7).getName());
    }

    @Test
    void testStationFailureRatesAreConfigured() {
        // All stations should have the same configured failure rate
        double expectedFailureRate = 0.002; // From application.properties

        for (Station station : stations) {
            assertEquals(
                    expectedFailureRate,
                    station.getStationFailureRate(),
                    "Station " + station.getName() + " should have configured failure rate");
        }
    }

    @Test
    void testAssemblyLineStationsAreSameAsInjectedList() {
        List<Station> assemblyStations = assemblyLine.getStations();

        // Verify same stations are used (but different list instance due to List.copyOf)
        assertEquals(stations.size(), assemblyStations.size());

        for (int i = 0; i < stations.size(); i++) {
            assertSame(
                    stations.get(i),
                    assemblyStations.get(i),
                    "Station " + i + " should be the same bean instance");
        }
    }
}



================================================================================
FILE: src\test\java\com\cu5448\pcb\config\SpringConfigurationTest.java
================================================================================

package com.cu5448.pcb.config;

import static org.junit.jupiter.api.Assertions.*;
```

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.TestConstructor;

import com.cu5448.pcb.controller.SimulationController;
import com.cu5448.pcb.factory.PCBFactory;
import com.cu5448.pcb.service.AssemblyLine;

import lombok.RequiredArgsConstructor;

@SpringBootTest
@TestConstructor(autowireMode = TestConstructor.AutowireMode.ALL)
@RequiredArgsConstructor
class SpringConfigurationTest {

    private final SimulationController simulationController;

    private final AssemblyLine assemblyLine;

    private final PCBFactory pcbFactory;

    private final StationProperties stationProperties;

    private final PCBProperties pcbProperties;

    @Test
    void testSpringDependencyInjection() {
        // Verify that all Spring beans are properly injected
        assertNotNull(simulationController);
        assertNotNull(assemblyLine);
        assertNotNull(pcbFactory);
    }

    @Test
    void testConfigurationProperties() {
        // Verify that configuration properties are loaded correctly
        assertEquals(0.002, stationProperties.getFailureRate(), 0.0001);

        // Test PCB defect rates from properties (using Lombok-generated getters)
        assertEquals(0.05, pcbProperties.getTestboard().getPlaceComponentsDefectRate(), 0.0001);
        assertEquals(0.002, pcbProperties.getSensorboard().getPlaceComponentsDefectRate(), 0.0001);
        assertEquals(0.004, pcbProperties.getGatewayboard().getPlaceComponentsDefectRate(), 0.0001);
    }

    @Test
    void testPCBFactoryWithConfiguration() {
        // Test that PCB factory creates boards with configuration-driven defect rates
        var testBoard = pcbFactory.createPCB("Test Board");
        assertEquals("TestBoard", testBoard.getType());
        assertEquals(0.05, testBoard.getDefectRate("PlaceComponents"), 0.0001);

        var sensorBoard = pcbFactory.createPCB("Sensor Board");
        assertEquals("SensorBoard", sensorBoard.getType());
        assertEquals(0.002, sensorBoard.getDefectRate("PlaceComponents"), 0.0001);
    }
}



================================================================================
FILE: src\test\java\com\cu5448\pcb\factory\PCBFactoryTest.java
================================================================================

package com.cu5448.pcb.factory;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import com.cu5448.pcb.config.PCBProperties;
import com.cu5448.pcb.model.*;

class PCBFactoryTest {

    private PCBFactory factory;

    @BeforeEach
    void setUp() {
        factory = new PCBFactory(new PCBProperties());
    }

    @Test
    void testCreateTestBoard() {
        PCB pcb = factory.createPCB("testboard");
        assertInstanceOf(TestBoard.class, pcb);
        assertEquals("TestBoard", pcb.getType());
    }
```

```
    @Test
    void testCreateSensorBoard() {
        PCB pcb = factory.createPCB("sensorboard");
        assertInstanceOf(SensorBoard.class, pcb);
        assertEquals("SensorBoard", pcb.getType());
    }

    @Test
    void testCreateGatewayBoard() {
        PCB pcb = factory.createPCB("gatewayboard");
        assertInstanceOf(GatewayBoard.class, pcb);
        assertEquals("GatewayBoard", pcb.getType());
    }

    @Test
    void testCreateWithAlternativeNames() {
        assertInstanceOf(TestBoard.class, factory.createPCB("test"));
        assertInstanceOf(SensorBoard.class, factory.createPCB("sensor"));
        assertInstanceOf(GatewayBoard.class, factory.createPCB("gateway"));
    }

    @Test
    void testCreateWithInvalidType() {
        assertThrows(IllegalArgumentException.class, () -> factory.createPCB("invalid"));
    }
}



================================================================================
FILE: src\test\java\com\cu5448\pcb\factory\StationFactoryTest.java
================================================================================

package com.cu5448.pcb.factory;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.TestConstructor;

import com.cu5448.pcb.station.*;

import lombok.RequiredArgsConstructor;

/** Test class for StationFactory implementation */
@SpringBootTest
@TestConstructor(autowireMode = TestConstructor.AutowireMode.ALL)
@RequiredArgsConstructor
class StationFactoryTest {

    private final StationFactory stationFactory;

    @Test
    void testFactoryInjection() {
        assertNotNull(stationFactory, "StationFactory should be injected");
    }

    @Test
    void testCreateIndividualStations() {
        // Test station creation using abstract factory method
        Station applySolderPaste = stationFactory.createStation("ApplySolderPaste");
        assertNotNull(applySolderPaste);
        assertEquals("ApplySolderPaste", applySolderPaste.getName());

        Station placeComponents = stationFactory.createStation("PlaceComponents");
        assertNotNull(placeComponents);
        assertEquals("PlaceComponents", placeComponents.getName());

        Station reflowSolder = stationFactory.createStation("ReflowSolder");
        assertNotNull(reflowSolder);
        assertEquals("ReflowSolder", reflowSolder.getName());

        Station opticalInspection = stationFactory.createStation("OpticalInspection");
        assertNotNull(opticalInspection);
        assertEquals("OpticalInspection", opticalInspection.getName());

        Station handSoldering = stationFactory.createStation("HandSoldering");
        assertNotNull(handSoldering);
        assertEquals("HandSoldering", handSoldering.getName());

        Station cleaning = stationFactory.createStation("Cleaning");
        assertNotNull(cleaning);
        assertEquals("Cleaning", cleaning.getName());

        Station depanelization = stationFactory.createStation("Depanelization");
        assertNotNull(depanelization);
        assertEquals("Depanelization", depanelization.getName());
```

```
        Station test = stationFactory.createStation("Test");
        assertNotNull(test);
        assertEquals("Test", test.getName());
    }

    @Test
    void testCreateStationByType() {
        // Test station creation by type name using abstract factory
        Station applySolder = stationFactory.createStation("ApplySolderPaste");
        assertInstanceOf(ApplySolderPasteStation.class, applySolder);
        assertEquals("ApplySolderPaste", applySolder.getName());

        Station placeComponents = stationFactory.createStation("PlaceComponents");
        assertInstanceOf(PlaceComponentsStation.class, placeComponents);
        assertEquals("PlaceComponents", placeComponents.getName());

        Station test = stationFactory.createStation("Test");
        assertInstanceOf(TestStation.class, test);
        assertEquals("Test", test.getName());
    }

    @Test
    void testCreateStationByTypeInvalid() {
        // Test invalid station type using abstract factory
        assertThrows(
                IllegalArgumentException.class,
                () -> stationFactory.createStation("InvalidStation"),
                "Should throw exception for invalid station type");

        assertThrows(
                IllegalArgumentException.class,
                () -> stationFactory.createStation(""),
                "Should throw exception for empty station type");

        assertThrows(
                IllegalArgumentException.class,
                () -> stationFactory.createStation("SomeRandomName"),
                "Should throw exception for random station type");
    }

    @Test
    void testAllStationTypesSupported() {
        // Test all expected station types are supported by abstract factory
        String[] stationTypes = {
            "ApplySolderPaste",
            "PlaceComponents",
            "ReflowSolder",
            "OpticalInspection",
            "HandSoldering",
            "Cleaning",
            "Depanelization",
            "Test"
        };

        for (String stationType : stationTypes) {
            assertDoesNotThrow(
                    () -> {
                        Station station = stationFactory.createStation(stationType);
                        assertNotNull(
                                station, "Station should be created for type: " + stationType);
                    },
                    "Should be able to create station for type: " + stationType);
        }
    }
}



================================================================================
FILE: src\test\java\com\cu5448\pcb\model\DefectRatesTest.java
================================================================================

package com.cu5448.pcb.model;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

/** Test class for DefectRates model */
class DefectRatesTest {

    @Test
    void testBuilderPattern() {
        DefectRates rates =
                DefectRates.builder()
                        .placeComponentsDefectRate(0.01)
                        .opticalInspectionDefectRate(0.02)
                        .handSolderingDefectRate(0.03)
                        .testDefectRate(0.04)
```

```
                        .build();

        assertEquals(0.01, rates.getPlaceComponentsDefectRate());
        assertEquals(0.02, rates.getOpticalInspectionDefectRate());
        assertEquals(0.03, rates.getHandSolderingDefectRate());
        assertEquals(0.04, rates.getTestDefectRate());
    }

    @Test
    void testGetDefectRateWithValidStations() {
        DefectRates rates =
                DefectRates.builder()
                        .placeComponentsDefectRate(0.05)
                        .opticalInspectionDefectRate(0.10)
                        .handSolderingDefectRate(0.05)
                        .testDefectRate(0.10)
                        .build();

        assertEquals(0.05, rates.getDefectRate("PlaceComponents"));
        assertEquals(0.10, rates.getDefectRate("OpticalInspection"));
        assertEquals(0.05, rates.getDefectRate("HandSoldering"));
        assertEquals(0.10, rates.getDefectRate("Test"));
    }

    @Test
    void testGetDefectRateWithInvalidStation() {
        DefectRates rates =
                DefectRates.builder()
                        .placeComponentsDefectRate(0.05)
                        .opticalInspectionDefectRate(0.10)
                        .handSolderingDefectRate(0.05)
                        .testDefectRate(0.10)
                        .build();

        assertEquals(0.0, rates.getDefectRate("ApplySolderPaste"));
        assertEquals(0.0, rates.getDefectRate("ReflowSolder"));
        assertEquals(0.0, rates.getDefectRate("Cleaning"));
        assertEquals(0.0, rates.getDefectRate("Depanelization"));
        assertEquals(0.0, rates.getDefectRate("NonExistentStation"));
    }

    @Test
    void testPCBIntegration() {
        // Test that PCB implementations can use DefectRates
        DefectRates testRates =
                DefectRates.builder()
                        .placeComponentsDefectRate(0.05)
                        .opticalInspectionDefectRate(0.10)
                        .handSolderingDefectRate(0.05)
                        .testDefectRate(0.10)
                        .build();
        TestBoard testBoard = new TestBoard(testRates);
        DefectRates defectRates = testBoard.getDefectRates();

        assertNotNull(defectRates);
        assertEquals(0.05, testBoard.getDefectRate("PlaceComponents"));
        assertEquals(0.05, defectRates.getDefectRate("PlaceComponents"));

        DefectRates sensorRates =
                DefectRates.builder()
                        .placeComponentsDefectRate(0.002)
                        .opticalInspectionDefectRate(0.002)
                        .handSolderingDefectRate(0.004)
                        .testDefectRate(0.004)
                        .build();
        SensorBoard sensorBoard = new SensorBoard(sensorRates);
        DefectRates actualSensorRates = sensorBoard.getDefectRates();

        assertNotNull(actualSensorRates);
        assertEquals(0.002, sensorBoard.getDefectRate("PlaceComponents"));
        assertEquals(0.002, actualSensorRates.getDefectRate("PlaceComponents"));
    }
}



================================================================================
FILE: src\test\java\com\cu5448\pcb\service\StatisticsCollectorTest.java
================================================================================

package com.cu5448.pcb.service;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class StatisticsCollectorTest {
```

```
    private StatisticsCollector stats;

    @BeforeEach
    void setUp() {
        stats = new StatisticsCollector();
    }

    @Test
    void testInitialState() {
        assertEquals(0, stats.getPcbsSubmitted());
        assertEquals(0, stats.getCompletedPCBs());
        assertTrue(stats.getDefectFailures().isEmpty());
        assertTrue(stats.getStationFailures().isEmpty());
    }

    @Test
    void testRecordSubmission() {
        stats.recordSubmission();
        stats.recordSubmission();
        assertEquals(2, stats.getPcbsSubmitted());
    }

    @Test
    void testRecordCompletion() {
        stats.recordCompletion();
        stats.recordCompletion();
        assertEquals(2, stats.getCompletedPCBs());
    }

    @Test
    void testRecordDefectFailure() {
        stats.recordDefectFailure("PlaceComponents");
        stats.recordDefectFailure("PlaceComponents");
        stats.recordDefectFailure("Test");

        assertEquals(2, stats.getDefectFailures().get("PlaceComponents"));
        assertEquals(1, stats.getDefectFailures().get("Test"));
    }

    @Test
    void testRecordStationFailure() {
        stats.recordStationFailure("ApplySolderPaste");
        stats.recordStationFailure("Cleaning");

        assertEquals(1, stats.getStationFailures().get("ApplySolderPaste"));
        assertEquals(1, stats.getStationFailures().get("Cleaning"));
    }

    @Test
    void testGenerateReport() {
        stats.recordSubmission();
        stats.recordSubmission();
        stats.recordCompletion();
        stats.recordDefectFailure("Test");
        stats.recordStationFailure("Cleaning");

        String report = stats.generateReport("Test Board");

        assertTrue(report.contains("PCB type: Test Board"));
        assertTrue(report.contains("PCBs run: 2"));
        assertTrue(report.contains("Total failed PCBs: 1"));
        assertTrue(report.contains("Total PCBs produced: 1"));
        assertTrue(report.contains("Test (ICT or Flying Probe) 1"));
        assertTrue(report.contains("Cleaning: 1"));
    }
}
```