Q1

```java
public abstract class ServiceTest {

	private Service service;
	protected abstract Service createService();

	@BeforeEach
	public void init() {
		service = this.createService();
	}

	@Test
	public void testIsUsedByWhenOk() {
		ServiceUser user = new ServiceUser(2* service.cost());
		int moneyBefore = user.getMoney();
		int nbUseBefore = service.getNumberOfUse();

		// Appel de la méthode à tester
		service.isUsedBy(user); // nbUse++ dans isUsedBy()

		assertEquals(1, service.getNumberOfUse() - nbUseBefore);
		assertEquals(service.cost(), moneyBefore - user.getMoney());
	}


	@Test
	public void testIsUsedByWhenUserHasNotEnoughMoney() {
		service = this.createService();
		ServiceUser user = new ServiceUser(-1);
		// ou bien : user.decreaseMoney(user.getMoney()-1);
		assertThrows(NotEnoughMoneyException.class, () -> {
			service.isUsedBy(user);
		});
	}

	// la création d'un mock "MockService" n'est pas ici nécessaire puisque
	// cette classe de test est abstraite et que ses tests sont exécutés via
	// les sous-classes qui doivent nécessairement définir createService()
}


public class ServiceOneTest extends ServiceTest {
	protected Service createService() {
		return new ServiceOne();
	}
}


public class ServiceOne extends Service {
	public int cost() {
		return 100;
	}

	public void execute() {
		// do something for ServiceOne
	}
}
```

Q2

```java
public class ProductionUnitTest {

	private ProductionUnit createProductionUnit() {
		return new MockUnit();
	}

	private ManufacturedProduct product;

	@BeforeEach
	public void init() {
		this.product = new Mockproduct();
	}

	@Test // Points 3 et 4 du cahier des charges (CdC)
	public void testOperateCountProductsWhenEnoughResource() {
		MockUnit mockUnit = this.createProductionUnit();
		mockUnit.addAvailableresource(100);
		mockUnit.operate(this.product); // le process de 1 produit requiert 10
		mockUnit.operate(this.product); // voir MockProduit plus bas
		assertEquals(2, mockUnit.getNbProcessedProducts());
		assertEquals(2,mockUnit.processCalled); // ← utilité du Mock
	}

	@Test
	public void testOperateCountProductsWhenNotEnoughResource() {
		MockUnit mockUnit = this.createProductionUnit();
		mockUnit.addAvailableresource(15);
		mockUnit.operate(this.product);
		mockUnit.operate(this.product);
		assertEquals(1, mockUnit.getNbProcessedProducts()); // Points 4 du CdC
		assertEquals(2, mockUnit.processCalled); // Points 3 du CdC
	}

	@Test
	public void testProcess() throws ProductionProblemException {
		MockUnit mockUnit = this.createProductionUnit();
		unit.addAvailableresource(100);
		int initialResources = unit.getAvailableResources();
		int initialDuration = this.product.getProductionDuration();
		unit.process(this.product);
		// Points 2.a du CdC
		assertEquals(this.product.requiredResourceQuantity(),
				initialResources-unit.getAvailableResources());
		// Points 2.b du CdC
		assertEquals(this.product.getProductionDuration(),
				initialDuration + unit.getProductionDuration());
	}

	@Test // Points 1 du CdC
	public void testProcessThrowsExceptionWhenNotEnoughResources() throws
					ProductionProblemException {
		MockUnit mockUnit = this.createProductionUnit();
		assertThrows(ProductionProblemException.class, () -> {
			unit.process(this.product);
		});
	}
```

```java
	// Parce que ProductionUnit est abstraite
	private class MockUnit extends ProductionUnit {
		public int processCalled = 0;

		public int getProductionDuration() {
			return 10;
		}

		public void process(ManufacturedProduct product) throws
					ProductionProblemException {
			this.processCalled ++;
			super.process(product);
		}

		protected void handleProductProcess(ManufacturedProduct product) {}

	}


	private class Mockproduct implements ManufacturedProduct {

		private int productionDuration = 0;
		public int requiredResourceQuantity() {
			return 10;
		}

		public int getProductionDuration() {
			return this.productionDuration;
		}

		public void increaseProductionDuration(int duration) {
			this.productionDuration += duration;
		}

		public void transformBy(ProductionUnit unit) {	}
	}
}
```