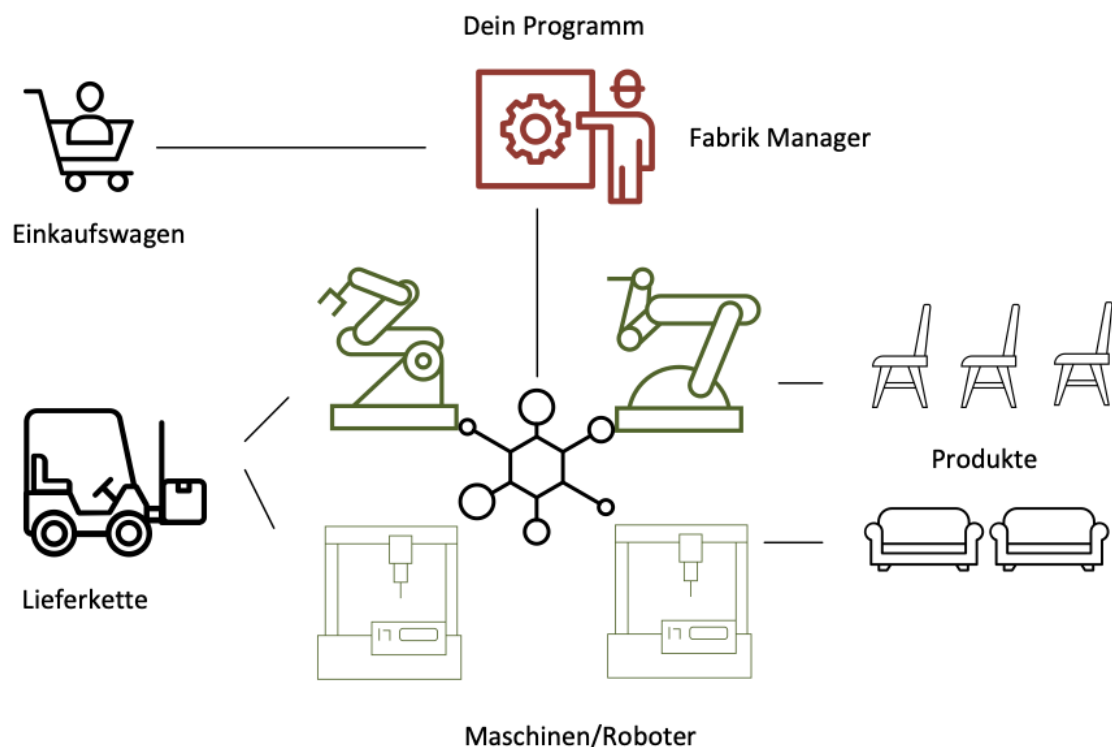


Ein Programmierprojekt in Smart Door Manufacturing

Business Case

Smart Manufacturing gilt als einer der Treiber der «4. Industriellen Revolution». Dank des Fortschrittes in den Fertigungstechnologien, müssen Maschinen und Montagelinien in einer Fabrik nicht mehr nur für ein bestimmtes Produkt gebaut werden, sondern können flexibel an sich ändernden Bedürfnissen und Lieferbedingungen angepasst werden. Aeki ist ein kostengünstiger, aber innovativer Türenhersteller, der ein Produktionsmanagementsystem für seine moderne Fabrik möchte. Die Fabrik ist mit Robotern, computergesteuerten Holzbearbeitungsmaschinen, Verpackungsstationen usw. ausgestattet. Es fehlt jedoch die Software, die die Produktion mit optimalem Ressourceneinsatz orchestriert und gleichzeitig die Produktlieferung sicherstellt. Der CEO von Aeki stellt Ihnen die Vision der Firma vor, die wir in der folgenden Abbildung zusammengefasst haben.



Einblick in das Geschäftsmodell

Ihr System ermöglicht die direkte Interaktion einer Kundenbestellschnittstelle mit der Fabrik. Ein klassischer Fall der Business-to-Customer-Interaktion (B2C), der eine hohe Flexibilität im Umgang mit Auftragsschwankungen erfordert. Die Auswahlmöglichkeiten rund um eine Türenbestellung basieren auf vorkonfigurierten Materialien und Produktionskapazitäten. Es wird auf Basis bekannter Türenarten und -mengen nach dem Prinzip der *Produktion auf Bestellung (Make-to-Order: MTO)* produziert.

Sie lernen und demonstrieren anhand eines Ökosystems von B2C- und MTO-Modellen, wie Softwaresysteme, insbesondere das objektorientierte Paradigma, skalierbare und erweiterbare Lösungen ermöglichen.

Deine Firma (Projektgruppe)

Als junges und dynamisches Startup im Bereich Smart Manufacturing möchten Sie eine Smart Factory Management Software entwickeln, die nicht nur die Anforderungen von Aeki erfüllt, sondern auch ein Design umsetzt, das sich gut mit der Entwicklung von Technologien skalieren lässt. Es gibt auch andere Startups, die versuchen, Aeki von den Vorteilen ihres jeweiligen Produktes zu überzeugen. Aber vorerst möchten Sie Aekis Problem lösen und auch etwas Geld verdienen.

Notizen aus einem technischen Meeting (User Story)

Sie treffen den Produktionsleiter von Aeki, der Ihnen erzählt, was sie erreichen wollen (*was* und nicht *wie*). In ihrer neuen Fabrik in St. Gallen will Aeki zwei Produkte produzieren – Standardtüren und Premiumtüren. Möglicherweise möchten sie später weitere Produkte hinzufügen. Die Herstellung kann mit vier automatisierten Maschinen durchgeführt werden – einer Holzbearbeitungs-CNC-Maschine und Montage-, Lackier- und Verpackungsrobotern. Es dauert 1 Stunde, um eine Maschine oder einen Roboter für die Handhabung eines bestimmten Produkttyps umzukonfigurieren – daher ist es keine gute Idee, die Produktion zwischen Türentypen häufig zu wechseln.

Die Holzarbeit bei der Türherstellung unterscheidet sich zwischen Standard- und Premiumtüren. Standardtüren benötigen 10 Minuten für das Zuschneiden von MDF oder Spanplatten, Fräsen und Bohren mit CNC-Maschinen. Premiumtüren erfordern 30 Minuten für die Verarbeitung von Massivholz wie Eiche oder Kirsche, einschließlich präzisiertem Zuschnitt, Profilfräsen und Handschleifen. Beide Prozesse enden mit einer Qualitätskontrolle und bereiten die Türen für die folgenden Fertigungsschritte vor. Weitere Herstellungsphasen (z.B. Montag, Spritzlackierung, Verpackung) sind **nicht** Gegenstand dieser Beschreibung.

Sie erhalten laufend Kundenaufträge. Sie müssen nach Rohstoffen suchen – dies sind:

- **Standardtür:** Holz (2 Einheiten), Schrauben (10 Einheiten), Farbe (2 Einheiten) und Karton (1 Einheit) für Verpackung
- **Premiumtür:** Holz (4 Einheiten), Schrauben (5 Einheiten), Glas (5 Einheiten), Farbe (1 Einheit) und Karton (5 Einheiten) für Verpackung

Sie haben nur begrenzten Lagerplatz in Ihrem Werk:

- Holz: 1000 Einheiten
- Schrauben: 5000 Einheiten
- Glas: 100 Einheiten
- Farbe: 1000 Einheiten
- Karton: 1000 Einheiten

Wenn Ihr Lagerbestand knapp wird, müssen Sie beim Lieferanten nachbestellen. Es dauert 2 Tage, bis eine Bestellung geliefert wird. Für jede Lieferung fallen feste Kosten an, also vermeiden Sie kleine Bestellungen.

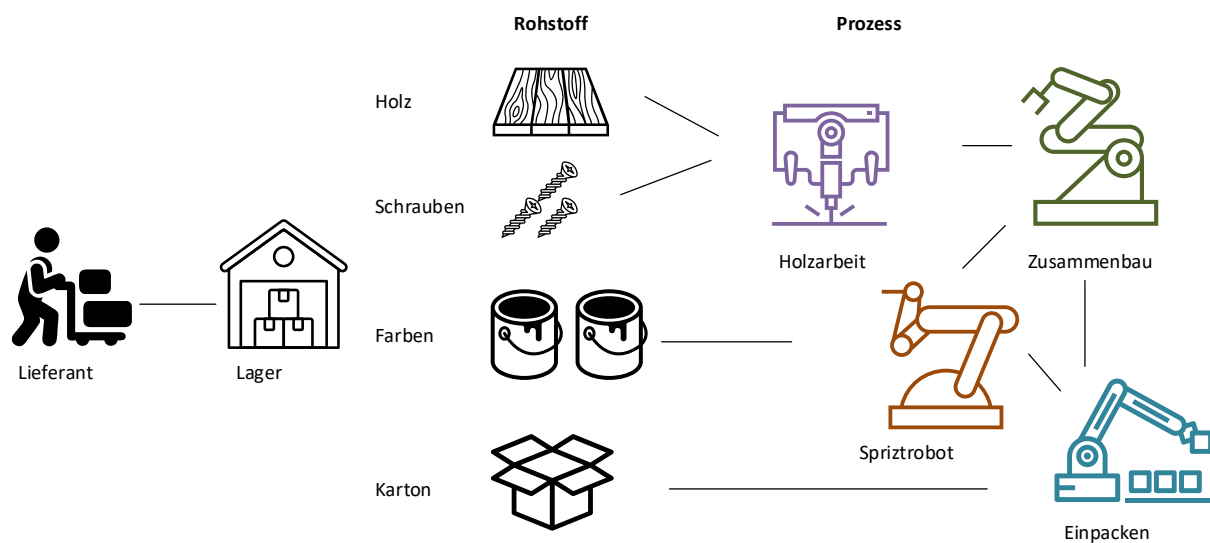
Optional: Bitte beachten Sie, dass sich Kosten, Lagerkapazitäten, Zahlen, Zeitpläne und Preise im Laufe der Zeit ändern können. Wir empfehlen daher, diese Informationen flexibel zu handhaben und nicht als feste Werte zu codieren. Unser Ziel ist es, unnötige Kommunikation zu vermeiden. Daher werden wir Sie nicht bei jeder dieser Änderungen kontaktieren. Es wäre hilfreich, wenn Ihr System solche Anpassungen automatisch verarbeiten könnte.

Nachdem Sie eine Türenbestellung erhalten haben, sollten Sie eine Vorlaufzeit angeben.

Ich sollte jederzeit wissen, was in der Fabrik vor sich geht – welche Maschine macht was, wie ist der Auftragsfortschritt.

Optional: Wenn das Geschäft gut läuft, ist Aeki bereit in Maschinen zu investieren. Aber Sie müssen der Firma die aktuelle Auslastung und die Engpässe mitteilen. Wenn eine neue Maschine hinzugefügt wird, sollte Ihre Produktionsmanagement-Software gleichbleiben. Es kann auch in Zukunft mehrere Lieferanten geben, die dynamische Preise anbieten, damit Sie Ihren Lagerbestand und Ihre Kosten optimieren können.

Natürlich können Maschinen und Roboter ausfallen, daher sollte Ihre Software mit solchen Bedingungen umgehen.



Aufgabe 0 – Anforderungsanalyse (0 Punkte)

Überlegen Sie, wie die Geschäftsziele von Aeki – Welche sind diese? – durch Ihre Individualsoftware unterstützt werden können. In welchem Teil des Geschäftsprozesses von Aeki spielen sie eine Rolle und wie können Sie Ihr Angebot in Zukunft darüber hinaus erweitern? Können Sie die Stakeholder identifizieren? Welche Bedürfnisse haben die Stakeholder und wie können Sie die Akzeptanzkriterien dieser ausdrücken?

Bereiten Sie sich vor, indem Sie das Geschäft und den Anwendungsfall analysieren. Nicht alles ist explizit (wie in der realen Welt), Sie müssen vielleicht fragen, annehmen, innovativ sein ... aber Sie können Aeki nicht fragen, wie Sie Ihre Software implementieren sollen! Denken Sie viel über den Problemraum nach (Sie werden überrascht sein, wie einfach die Lösung sein kann). Seien Sie kreativ!

Nachdem Sie die Anwendungsfälle verstanden haben, schreiben Sie ein einseitiges Anforderungsdokument, in dem angegeben ist, was Ihre Software leisten soll.

Die Aufgabe 0 wird nicht bewertet. Organisieren Sie im Zuge der Aufgabe 0 Ihre Gruppe und denken Sie über Anforderungen und Projektorganisation nach. Vieles davon müssen Sie in den folgenden Aufgaben wiederverwenden.

Punkte: Muss eingereicht werden, wird aber nicht bewertet

Fällig: 22.09.2024 (23:59 CET)

Einzureichen:

- Einseitiges Anforderungsdokument, das die funktionalen und nichtfunktionalen Anforderungen des Programms enthält. Die Anforderungen können kurze Aussagen sein.

Zu beachten:

- Anforderungserklärungen decken die User-Story ab
- Sie haben einige grundlegende nichtfunktionale Anforderungen identifiziert

Aufgabe 1 – Implementierung 1 (10 Punkte)

Als Grundlage für diese Implementierungsaufgabe stehen Ihnen die Musterlösung des Klassendiagramms für diese Teilaufgabe und ein ausgewähltes Sequenzdiagramme zur Verfügung. Versuchen Sie diese Diagramme zu verstehen, denn sie dienen als Grundlage für Ihre erste Programmieraufgabe.

In der Implementierung möchten wir, dass Sie eine Klasse *Fabrik* implementieren, die *Bestellungen* entgegennimmt und diese auf der Konsole ausgibt. Die Klasse *Fabrik* enthält als globale Variable eine *ArrayList* in der alle Bestellungen gespeichert werden. Mit der Methode *bestellungAufgeben* ist es möglich eine neue *Bestellung* zu platzieren, in der die Anzahl bestellter *Standardtüren* und *Premiumtüren* angegeben werden. Die Methode *bestellungenAusgeben* durchläuft die Liste der Bestellungen und gibt jeweils die Information zu den einzelnen Bestellungen auf der Konsole aus. Die Klasse *Fabrik* enthält die Methode *main*, die den Einstieg in das Programm ermöglicht.

Die Klasse *Bestellung* verwaltet eine *ArrayList*, in der alle bestellten *Produkte* gespeichert werden. *Produkte* können entweder *Standardtüren* oder *Premiumtüren* sein. Als globale Variablen enthält die Klasse *Bestellung* die *Bestellbestätigung*, *Beschaffungszeit*, *Bestellnummer*, wie auch die *Anzahl bestellter Standardtüren* und die *Anzahl bestellter Premiumtüren*. Zu jeder globalen Variablen muss jeweils eine Methode implementiert werden, um die Information abzufragen. Bei gewissen globalen Variablen muss auch eine Methode vorhanden sein, um die Information in den Variablen zu ändern.

Die Klassen *Standardtür* und *Premiumtür* sind Erweiterungen von der Klasse *Produkt* und erben somit die Funktionalität und die globalen Variablen der Klasse *Produkt*. Jedes Produkt befindet sich in einem gewissen *Zustand*, beispielsweise *bestellt*, *in Produktion* etc.. Dieser könnte auch in einer globalen Variablen abgespeichert werden. Für die Klasse *Standardtür* und *Premiumtür* sind Information vorhanden, die für alle Instanzen dieser Klassen gleich sind. Beispielsweise wären das die *Anzahl Holzeinheiten*, *Anzahl Schrauben*, etc. Auch die Methoden, um auf diese Information zuzugreifen, gehören der Klasse an und nicht den einzelnen Instanzen.

Natürlich sind noch weitere Methoden und Variablen notwendig, um das Programm zum Laufen zu bringen. Auch eine Testklasse für die Fabrik wird benötigt, um Bestellungen aufzugeben und die bestellten Produkte ausgeben.

Punkte: 10

Abgabedatum: 27.10.2024 (23:59 CET)

Einzureichen:

- Implementierter und funktionsfähiger Code
- Testklasse
- Ein zip-File mit Ihrem gesamten BlueJ-Projekt

Bewertungskriterien:

- Die Qualität Ihrer Codeimplementierung – Strukturierung, Benennung, Kommentare, Fehlerbehandlung – **2 Punkte**
- Die richtige Funktionsweise des implementierten Codes – **5 Punkte**
- Genügend Abdeckung der implementierten Tests, um die Funktionalität des Codes vollständig überprüfen zu können – **3 Punkte**

Aufgabe 2 – Implementierung 2 (5 Punkte)

Nachdem Sie die Bestellverwaltung programmiert haben, erweitern wir nun das Aeki-Programm mit folgender Funktionalität: Zu jeder Bestellung muss nun eine Auftragsbestätigung erfolgen. Vor der Rücksendung einer Auftragsbestätigung prüft die Fabrik mit dem Lager, ob ausreichende Lagerbestände für die Fertigung der Bestellung vorhanden sind. Wir gehen in unserem Programm von einem **Make-To-Order**-Vorgang aus. Bei geringen Lagerbeständen veranschlagt die Auftragsbestätigung zwei Tage zusätzlich zur Standardlieferzeit, welche Sie mit einem Tag festlegen können. Die Lieferzeit errechnet sich durch die Produktionszeit der bestellten Türen plus die Beschaffungszeit der notwendigen Materialien plus die Standardlieferzeit. Wenn die Bestände niedrig sind, sollte das Lager eine Bestellung beim *Lieferanten* aufgeben können.

Das erstellte Programm wird somit um zwei weitere Klassen erweitert. Implementieren Sie die Klassen *Lager* und die Klasse *Lieferant*. Auch für diese Aufgabe stellen wir Ihnen das dazugehörige Klassendiagramm zur Verfügung.

1. Die Klasse *Lager* beinhaltet die Information zu den maximal lagerbaren Materialeinheiten. Diese sind in den Klassenvariablen *maxHolzeinheiten*, *maxSchrauben*, *maxFarbeinheiten*, *maxKartoneinheiten* und *maxGlaseinheiten* gespeichert.

Implementieren Sie in der Klasse *Lager* folgende Methoden:

- *gibBeschaffungszeit*

Die Methode erhält als Parameter eine Kundenbestellung und liefert *0 Tage* zurück, wenn alle Materialien für die Produktion aller bestellter Produkte vorhanden sind, und *2 Tage*, wenn das Material beim Lieferanten nachbestellt werden muss. Dafür muss die Liste mit allen Produkten der Bestellung durchsucht und die Anzahl benötigter Materialien ausgerechnet werden. Für die Berechnung müssen Sie wissen, ob es sich um eine Standardtür oder um eine Premiümtür handelt. Mit folgender Abfrage könne Sie dies erfahren:

```
if(produkt instanceof Standardtuer){  
    ...  
}else if(produkt instanceof Premiümtuer){  
    ...  
}
```

Die Variable *product* ist vom Typ *Produkt*.

- *lagerAuffuellen*

Diese Methode bestellt fehlende Produkte beim Lieferanten nach und füllt nach Erhalt das Lager wieder auf.

- *lagerBestandAusgeben*

Diese Methode druckt die im Lager vorhandenen Materialeinheiten auf die Konsole aus.

2. Die Klasse *Lieferant* besitzt nur eine Methode, welche der Fabrik ermöglicht, eine Bestellung aufzugeben.

Die Klassen *Lager* und *Lieferant* müssen auch noch erzeugt werden. Überlegen Sie sich genau, wo diese Klassen instanziiert werden müssen. Auch sind Anpassungen in den Klassen *Bestellung* und *Fabrik* notwendig.

In der Klasse *Bestellung* muss zu jeder Instanz auch die korrekte *Beschaffungszeit* und die entsprechende *Lieferzeit* gesetzt werden können. Eine Methode, welche die Liste mit allen bestellten Produkten einer Bestellung retourniert, wird auch in der Klasse *Bestellung* benötigt.

In der Klasse *Fabrik* ändert sich die Methode *bestellungAufgeben*, indem die *Beschaffungszeit* und die *Lieferzeit* errechnet und in der jeweiligen Bestellung gespeichert werden müssen. Irgendwann muss auch das Lager wieder aufgefüllt werden.

Erstellen Sie die Testklasse für Ihre Fabrik so, dass mehrere Bestellungen aufgegeben und die Auftragsbestätigungen überprüft werden können. Testen Sie auch, ob das Lager eine Bestellung an den Lieferanten senden kann.

Punkte: 5

Abgabedatum: 24.11.2024 (23:59 CET)

Einzureichen:

- Implementierter und funktionsfähiger Code
- Testklasse
- Ein zip-File mit Ihrem gesamten BlueJ-Projekt

Bewertungskriterien:

- Die Qualität Ihrer Codeimplementierung – Strukturierung, Benennung, Kommentare, Fehlerbehandlung – **1 Punkt**
- Die richtige Funktionsweise des implementierten Codes – **3 Punkte**
- Genügend Abdeckung der implementierten Tests, um die Funktionalität des Codes vollständig überprüfen zu können – **1 Punkt**

Aufgabe 3 – Implementierung 3 (15 Punkte)

Setzen Sie die vorherige Aufgabe fort und erweitern Sie die Klasse *Lieferant* so, dass die bestellten Teile erst in zwei Tagen an das Lager geliefert werden. Natürlich werden wir in unserer Software nicht zwei Tage warten; lassen Sie uns die Zeit beschleunigen und gehen Sie davon aus, dass eine Stunde einer Sekunde entspricht (1 Stunde = 1 Sekunde).

Um dies zu ermöglichen, müssen Sie die Klasse *Lieferant* als *Thread* implementieren.

Benutzen Sie die Testklasse der Fabrik, damit Sie überprüfen können, ob Ihre Bestellungen erst nach der definierten Zeit geliefert werden.

Auch für diese Aufgabe stellen wir Ihnen die Musterlösung des Klassendiagramms zur Verfügung.

Wenn die Implementierung der Klasse *Lieferant* als *Thread* funktioniert, fahren Sie folgendermassen fort: Die Fabrik übergibt nun die Bestellungen dem *Produktionsmanager*. Implementieren Sie dazu die Klasse *Produktions_Manager*. Diese Klasse sollte auch als *Thread* implementiert werden, damit sie immer wieder neu eintreffende Bestellungen abarbeiten und den Robotern zum Produzieren geben kann. Im *Konstruktor* der Klasse *Produktions_Manager* sollte exemplarisch der *Holzbearbeitungs_Roboter* als *Thread* instanziiert und gestartet werden. Die Roboter *Montage_Roboter*, *Lackier_Roboter* und *Verpackungs_Roboter* sind **nicht** zu implementieren. Auch sind zwei *LinkedLists* notwendig, um die zu verarbeitende Bestellungen und die Bestellungen in *Produktion* zu verwalten. In der Klasse *run* ist eine unendliche Schleife notwendig.

```
while(true){
    //Ist eine neue Bestellung eingetroffen, dann
    //hole die nächste Bestellung und starte die Produktion
    ...
    //dann lass den Thread eine kurze Weile schlafen
    try{
        Thread.sleep(zeit);
    }catch (InterruptedException ie){
        ie.printStackTrace();
    }
}
```

Diese unendliche Schleife prüft immer wieder, ob eine neue Bestellung eingetroffen ist. Diese Bestellung wird dann aus der Liste der zu verarbeitenden Bestellungen rausgenommen und in die Liste der zu produzierenden Bestellungen gespeichert. Dann wird die Produktion gestartet.

Für die Implementierung der Methoden *starteProduktion* und *naechsteProduktionsStation* in dieser Programmierübung liegt der Fokus ausschließlich auf dem *Holzbearbeitungs_Roboter*. Die Methode *starteProduktion* erhält eine Bestellung als Parameter und alloziert den *Holzbearbeitungs_Roboter* für jedes Produkt der Bestellung. Dabei ist zu beachten, dass die Bearbeitungszeit für Standardtüren 10 Minuten und für Premiumtüren 30 Minuten beträgt. Die Allokation des *Holzbearbeitungs_Roboters* wird in einer Liste im Produkt-Objekt gespeichert.

Die Methode *naechsteProduktionsStation* in der Klasse *Produkt* extrahiert den *Holzbearbeitungs_Roboter* aus dieser Liste und fügt das aktuelle Produkt zur Produktionsliste des Roboters hinzu. Andere Roboter und Produktionsschritte werden in dieser Übung nicht implementiert, sind jedoch für den vollständigen Produktionsprozess konzeptionell zu berücksichtigen.

Für die Produktion müssen Sie die Klasse *Holzbearbeitungs_Roboter* implementieren. Diese ist eine Erweiterung der Klasse *Roboter*, welche wiederum eine Erweiterung der Klasse *Thread* ist. Die Klassen *Montage_Roboter*, *Lackier_Roboter* und *Verpackungs_Roboter* sind **nicht** zu implementieren

Die Klasse *Roboter* verwaltet eine Liste (Warteschlange) mit allen zu produzierenden Produkten. In der Methode *run* ist wieder eine unendliche Schleife notwendig, um immer wieder zu schauen, ob neue Produkte, welche produziert werden müssen, eingetroffen sind. Ist ein Produkt in der Warteschlange eingetroffen, so produziert der Roboter dieses Produkt.

In der Methode *produziereProdukt* wird die Holzbearbeitung durch den *Holzbearbeitungs_Roboter* simuliert. Der Thread wird für 10 Minuten (600.000 Millisekunden) bei Standardtüren oder 30 Minuten (1.800.000 Millisekunden) bei Premiumtüren schlafen gelegt, um die Bearbeitungszeit zu simulieren. Nach Abschluss der Holzbearbeitung wird das Produkt als bereit für den nächsten Produktionsschritt markiert, wobei die Implementierung weiterer Roboter oder Stationen in dieser Übung nicht erforderlich ist. Die Methode sollte robust gegenüber *InterruptedException* sein und eine angemessene Fehlerbehandlung beinhalten.

Im *Produktions_Manager* wird geprüft, ob eine Bestellung abgeschlossen ist. Wenn ja, wird dies in der Klasse *Bestellung* vermerkt und eine entsprechende Nachricht im Terminal ausgegeben.

Beachten Sie, dass sich die Lagerbestände während der Produktion verringern und bei Bedarf nachbestellt werden müssen. Dafür sollte die Klasse *Lager* um diese neue Funktionalität ergänzt werden. Wir gehen davon aus, dass wir nie einen Kundenauftrag erhalten, der Rohstoffe benötigt, die die Lagerkapazität überschreiten.

Damit der Ablauf in der Fabrik nachvollzogen werden kann, verwenden Sie das Terminal, um jeweils Meldungen auszugeben.

Erweitern Sie die Testklasse der Fabrik, wenn nötig, damit Sie überprüfen können, ob Ihre Bestellungen geliefert werden.

Punkte: 15

Abgabedatum: 15.12.2024 (23:59 CET)

Einzureichen:

- Implementierter und funktionsfähiger Code
- Testklasse
- Ein zip-File mit Ihrem gesamten BlueJ-Projekt

Bewertungskriterien:

- Implementierung Ihres Produktionsmanagement- und Lieferantenbestellprozesses, der zeigt, wie Sie die Bestellungen abarbeiten und die Produkte produzieren – **8 Punkte**
- Die richtige Funktionsweise des implementierten Codes – **2 Punkte**
- Die Qualität Ihrer Codeimplementierung – Strukturierung, Benennung, Kommentare, Fehlerbehandlung – **2 Punkte**
- Genügend Abdeckung der implementierten Tests, um die Funktionalität des Codes vollständig überprüfen zu können – **3 Punkte**

Aufgabenübersicht

Aufgabe	Inhalt	Punkte	Abgabedatum
Aufgabe – 0	Anforderungsanalyse	0	22.09.24 23:59 CET
Aufgabe – 1	Implementierung – Teil 1	10	27.10.24 23:59 CET
Aufgabe – 2	Implementierung – Teil 2	5	24.11.24 23:59 CET
Aufgabe – 3	Implementierung – Teil 3	15	15.12.24 23:59 CET
Total		30	