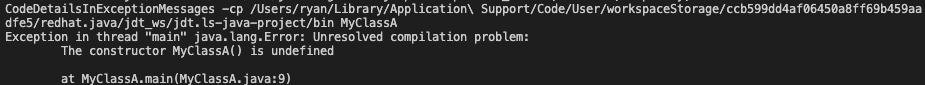
Homework 1, CMSC 335 6380

Ryan Gant, Prof Suzanna Schmeelk

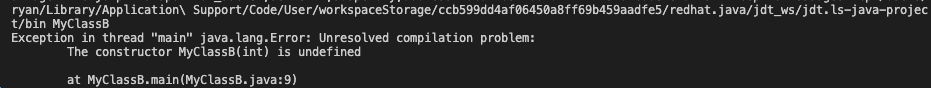
Aug 26, 2025

1.)



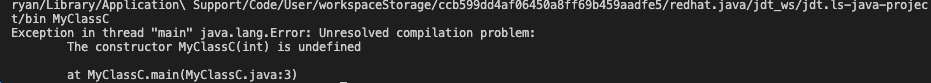
The error stems from the absence of a no-argument constructor for MyClassA, despite the code attempting to invoke one. To address this, one could either implement a default (no-parameter) constructor or modify the main method to supply an integer argument during instantiation. If the default constructor is added, the variable v retains its initialized value of 12. Alternatively, if the integer-argument constructor is used, v assumes the value provided as an argument.

2.)



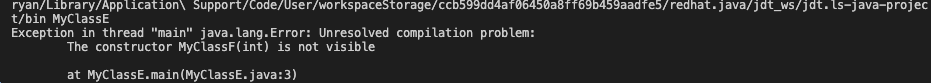
The error in this question is because constructors in Java must have the exact same name as the class and should not specify a return type. The code incorrectly defines the constructor as public void MyClassB(int pV), which is treated as a method, not a constructor. By removing the void keyword, the definition becomes a proper constructor, allowing v to be assigned the value of the argument passed in.

3.)



The logical error occurs when the constructor declares a new local variable v within its scope, thereby shadowing the instance variable. As a result, the instance variable is never actually updated. Eliminating the int keyword in the constructor ensures that the assignment targets the instance variable. After this correction, v holds the value provided as an argument.

4.)



The error arises from the constructor being declared private, which prevents external instantiation of the class. To resolve this, the constructor should be declared public, thus allowing the main method to create an instance of the class. Upon this modification, v is assigned the value of the provided argument.

5.)

The updates in Question 5 are effective because it introduces two explicitly defined constructors one accepting a boolean parameter and the other an integer. When the main method invokes the constructor with the arguments (23, true), it can identify and utilize the appropriate constructor, thereby ensuring the correct initialization of the instance variable v. This approach eliminates ambiguity in constructor selection and prevents errors related to undefined or inaccessible constructors, resulting in consistent and reliable behavior within the code.

6.)

• DefenseDepartment

o General

▪ Private

This class structure is not reasonable as “General” is an officer rank across the U.S militaries branches (excluding navy) and “Private” is an enlisted rank. Another issue with this hierarchy is rank names differ across different military branches for example, Navy’s “General” ranks are referred to as “Admirals” and branches such as the Air Force, Space Force, and Navy do not use the rank “Private”. What would be an appropriate structure would be:

• DefenseDepartment

o Army

▪ Officer

▪ General (O-7 – O11)

▪ Enlisted

▪ Private (E-1)

7.)

• Vehicle

o Car

* isElectric

o Airplane

▪ Passenger

* passengerCapacity

▪ Fighter

* isDogfighter

▪ Bomber

* isStealth

o Spaceship

* orbitalPayload

8.)

• Vehicle

o Car

* EngageReversalMotors()

o Airplane

▪ Passenger

* LockCargoDoors()

▪ Fighter

* EngageArielManuverFlaps()

▪ Bomber

* EnableStealth()

o Spaceship

* DeployPayload()

9.)

AirForceMember

* Name
* Rank
* AFSC
* CyberDefenseOperations(inherits from AirForceMember)
* ClearanceLevel
* CompleteTraining
* Mission
* MissionStartDate
* MissionEndDate

AirForceMember

* isPromoteable()
* assignAFSC()
* assignedDutyStation()
* CyberDefenseOperations(inherits from AirForceMember)
* verifyClearance()
* completedTrainingVerification()
* Mission
* startMission()
* endMission()

The AirForceMember class establishes fundamental attributes such as name, rank, and AFSC, serving as the essential basis for further specialization. Building upon this, the CyberDefenseOperations class extends these core properties, incorporating additional fields like ClearanceLevel and CompleteTraining, and utilizes the AFSC to confirm a member’s relevance to the cyber defense sphere. The Mission class, in turn, interacts with these structures by assigning tasks exclusively to individuals whose AFSC aligns with mission needs, who have fulfilled training requirements, and who possess the requisite clearance. Collectively, this framework exemplifies the Air Force’s systematic approach to personnel organization, role specialization, and the thorough validation of qualifications prior to mission assignment.

10.)

* JapaneseTrees
* DeciduousJapaneseTree
  + - * SakuraTree
      * MomijiMapleTree
* ConiferousJapaneseTree
  + - * MatsuPineTree
      * SugiCedarTree

The structure here is quite effective. The JapaneseTree base class encapsulates attributes common to all trees, serving as a foundation. Its subclasses, Deciduous and Coniferous, represent a biologically accurate division, much like the classifications found in botany. Specific types, such as SakuraTree and MatsuPineTree, inherit appropriate characteristics from their respective parent classes, while retaining space for distinct behaviors when necessary. This approach reduces redundancy and enhances clarity, faithfully mirroring real-world taxonomic hierarchies.