

Answer Key

1. There are two: (String, double) and (String). Java picks the one whose parameter list matches the arguments.
2. Static: numAccounts (private), interestRate (private), MIN_BALANCE (public). Instance: owner, balance, accountNumber (all private), and nickname (public).
3. MIN_BALANCE is a constant that is shared by all accounts and never changes. accountNumber is set once in the constructor and can never be reassigned.
4. There are three instance getters: getOwner, getBalance, and getAccountNumber, plus the static getNumAccounts. There is one instance setter, setOwner, plus the static setInterestRate. Follow-up: deposit and withdraw also change the object's state, so are they mutators too?
5. computeInterest. Only code inside BankAccount can call it.
6. getNumAccounts, setInterestRate, and main. Call them with the class name, for example BankAccount.getNumAccounts().
7. Both constructors use numAccounts, withdraw uses MIN_BALANCE, computeInterest uses interestRate, and getNumAccounts and setInterestRate use their own fields.
8. The only difference is the starting balance: startingBalance in one, 0.0 in the other. Setting the owner, setting the nickname, incrementing numAccounts, and assigning accountNumber are all duplicated. If someone edits one constructor (say, removes numAccounts++ or adds a new field) and forgets the other, the two kinds of accounts will behave differently, and the account count or numbering could break.
9. It returns 3, and the third account gets number 3.
10. Nothing changes and the method returns false, because the balance would drop below MIN_BALANCE.
11. Nothing happens. The if ignores amounts that aren't positive.
12. a has \$105. b is still \$50, because addInterest was only called on a. The rate change affects everyone, but interest is only added when addInterest() is called on that account.
13. It prints something like Account #1 (Ada): \$100.00, using a's actual values. println calls toString() automatically.
14. Inside the constructor, owner by itself refers to the parameter, so owner = owner assigns the parameter to itself and the field stays null. this.owner refers to the field.
15. It forces every change to go through deposit and withdraw, which enforce the rules. Nobody can just set the balance to a million dollars.
16. It's final, so nobody can change it. Reading it is harmless.
17. a.nickname = "anything"; Any code anywhere can set it to any value, including null, and the class has no control over that.
18. A static method isn't attached to any one object, so there's no this and no particular account's balance to return.
19. It compares accountNumber. The two accounts would have different numbers, so they are not equal. Discussion: for bank accounts, identity matters more than contents.
20. == checks whether the two variables refer to the same object; .equals uses the class's own rule. Two separate objects with the same account number would give == false but .equals true. That can't normally happen here because every constructor call makes a new number, so ask students why.

- 21.** The first account would get number 0, the second 1, and so on. If only one constructor were changed, the two kinds of accounts could even end up with duplicate numbers.
- 22.** Yes, `main` can call it, because `main` is inside the class (it needs an object, as in `a.computeInterest()`). A different class could not call it.

Watch for: #12 (static rate vs. calling the method on one object), #19 (what "equal" should mean), and #22 (private means private to the class, not to the object).