ANSWER KEY

1. **D**
2. **C**
3. **D**
4. **E**
5. **C**
6. **B**
7. **E**
8. **D**
9. **D**
10. **B**
11. **A**
12. **A**
13. **B**
14. **D**
15. **A**
16. **A**
17. **D**
18. **E**
19. **B**
20. **D**

ANSWERS EXPLAINED

1. **(D)** The methods are `deposit`, `withdraw`, and `getBalance`, all inherited from the `BankAccount` class, plus `addInterest`, which was defined just for the class `SavingsAccount`.

2. **(C)** Implementation I fails because `super()` *must* be the first line of the implementation whenever it is used in a constructor. Implementation III may appear to be incorrect because it doesn't initialize `interestRate`. Since `interestRate`, however, is a primitive type—`double`—the compiler will provide a default initialization of `0`, which was required.

3. **(D)** First, the statement `super(acctBalance)` initializes the inherited private variable `balance` as for the `BankAccount` superclass. Then the statement `interestRate = rate` initializes `interestRate`, which belongs uniquely to the `SavingsAccount` class. Choice E fails because `interestRate` does not belong to the `BankAccount` class and therefore cannot be initialized by a `super` method. Choice A is wrong because the `SavingsAccount` class cannot directly access the private instance variables of its superclass. Choice B assigns a value to an accessor method, which is meaningless. Choice C is incorrect because `super()` invokes the *default* constructor of the superclass. This will cause `balance` of the `SavingsAccount` object to be initialized to `0`, rather than `acctBalance`, the parameter value.

4. **(E)** The constructor must initialize the inherited instance variable `balance` to the value of the `acctBalance` parameter. All three segments achieve this. Implementation I does it by invoking `super(acctBalance)`, the constructor in the superclass. Implementation II first initializes `balance` to `0` by invoking the *default* constructor of the superclass. Then it calls the inherited `deposit` method of the superclass to add `acctBalance` to the account. Implementation III works because `super()` is automatically called as the first line of the constructor code if there is no explicit call to `super`.

5. **(C)** First the `withdraw` method of the `BankAccount` superclass is used to withdraw `amount`. A prefix of `super` must be used to invoke this method, which eliminates choices B and D. Then the balance must be tested using the accessor method `getBalance`, which is inherited. You can't test `balance` directly since it is private to the `BankAccount` class. This eliminates choices A and E, and provides another reason for eliminating choice B.

6. **(B)** When a superclass method is redefined in a subclass, the process is called *method overriding*. Which method to call is determined at run time. This is called *dynamic binding* (p. 146). *Method overloading* is two or more methods with different signatures