Questions 1–10 refer to the BankAccount, SavingsAccount, and CheckingAccount classes defined below:

```
public class BankAccount
{
    private double balance;

    public BankAccount()
    { balance = 0; }

    public BankAccount(double acctBalance)
    { balance = acctBalance; }

    public void deposit(double amount)
    { balance += amount; }

    public void withdraw(double amount)
    { balance -= amount; }

    public double getBalance()
    { return balance; }
}

public class SavingsAccount extends BankAccount
{
    private double interestRate;

    public SavingsAccount()
    { /* implementation not shown */ }

    public SavingsAccount(double acctBalance, double rate)
    { /* implementation not shown */ }

    public void addInterest()    //Add interest to balance
    { /* implementation not shown */ }
}

public class CheckingAccount extends BankAccount
{
    private static final double FEE = 2.0;
    private static final double MIN_BALANCE = 50.0;

    public CheckingAccount(double acctBalance)
    { /* implementation not shown */ }

    /** FEE of $2 deducted if withdrawal leaves balance less
     *  than MIN_BALANCE. Allows for negative balance. */
    public void withdraw(double amount)
    { /* implementation not shown */ }
}
```

1. Of the methods shown, how many different nonconstructor methods can be invoked by a `SavingsAccount` object?
   (A) 1
   (B) 2
   (C) 3
   (D) 4
   (E) 5

2. Which of the following correctly implements the default constructor of the `SavingsAccount` class?

```
  I interestRate = 0;
    super();

 II super();
    interestRate = 0;

III super();
```

   (A) II only
   (B) I and II only
   (C) II and III only
   (D) III only
   (E) I, II, and III

3. Which is a correct implementation of the constructor with parameters in the `SavingsAccount` class?

```
(A) balance = acctBalance;
    interestRate = rate;

(B) getBalance() =  acctBalance;
    interestRate = rate;

(C) super();
    interestRate = rate;

(D) super(acctBalance);
    interestRate = rate;

(E) super(acctBalance, rate);
```

4. Which is a correct implementation of the `CheckingAccount` constructor?

```
  I super(acctBalance);

 II super();
    deposit(acctBalance);

III deposit(acctBalance);
```

   (A) I only
   (B) II only
   (C) III only
   (D) II and III only
   (E) I, II, and III

5. Which is correct implementation code for the `withdraw` method in the `CheckingAccount` class?

   (A)
   ```
   super.withdraw(amount);
   if (balance < MIN_BALANCE)
       super.withdraw(FEE);
   ```
   (B)
   ```
   withdraw(amount);
   if (balance < MIN_BALANCE)
       withdraw(FEE);
   ```
   (C)
   ```
   super.withdraw(amount);
   if (getBalance() < MIN_BALANCE)
       super.withdraw(FEE);
   ```
   (D)
   ```
   withdraw(amount);
   if (getBalance() < MIN_BALANCE)
       withdraw(FEE);
   ```
   (E)
   ```
   balance -= amount;
   if (balance < MIN_BALANCE)
       balance -= FEE;
   ```

6. Redefining the `withdraw` method in the `CheckingAccount` class is an example of
   (A) method overloading.
   (B) method overriding.
   (C) downcasting.
   (D) dynamic binding (late binding).
   (E) static binding (early binding).

Use the following for Questions 7–9.
A program to test the `BankAccount`, `SavingsAccount`, and `CheckingAccount` classes has these declarations:

```
BankAccount b = new BankAccount(1400);
BankAccount s = new SavingsAccount(1000, 0.04);
BankAccount c = new CheckingAccount(500);
```

7. Which method call will cause an error?
   (A) `b.deposit(200);`
   (B) `s.withdraw(500);`
   (C) `c.withdraw(500);`
   (D) `s.deposit(10000);`
   (E) `s.addInterest();`

8. In order to test polymorphism, which method must be used in the program?
   (A) Either a `SavingsAccount` constructor or a `CheckingAccount` constructor
   (B) `addInterest`
   (C) `deposit`
   (D) `withdraw`
   (E) `getBalance`

9. Which of the following will *not* cause a `ClassCastException` to be thrown?
   (A) `((SavingsAccount) b).addInterest();`
   (B) `((CheckingAccount) b).withdraw(200);`
   (C) `((CheckingAccount) c).deposit(800);`
   (D) `((CheckingAccount) s).withdraw(150);`
   (E) `((SavingsAccount) c).addInterest();`

10. A new method is added to the `BankAccount` class.

```
/** Transfer amount from this BankAccount to another BankAccount.
 *  Precondition: balance > amount
 *  @param another a different BankAccount object
 *  @param amount the amount to be transferred
 */
public void transfer(BankAccount another, double amount)
{
    withdraw(amount);
    another.deposit(amount);
}
```

A program has these declarations:

```
BankAccount b = new BankAccount(650);
SavingsAccount timsSavings = new SavingsAccount(1500, 0.03);
CheckingAccount daynasChecking = new CheckingAccount(2000);
```

Which of the following will transfer money from one account to another without error?

I `b.transfer(timsSavings, 50);`

II `timsSavings.transfer(daynasChecking, 30);`

III `daynasChecking.transfer(b, 55);`

(A) I only
(B) II only
(C) III only
(D) I, II, and III
(E) None

11. Consider these class declarations:

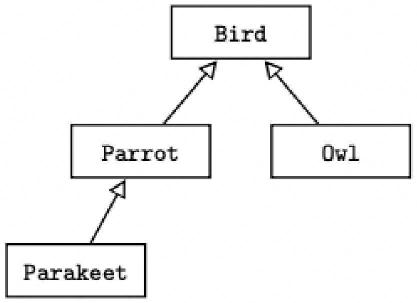
```
public class Person
{
    ...
}

public class Teacher extends Person
{
    ...
}
```

Which is a true statement?

I `Teacher` inherits the constructors of `Person`.
II `Teacher` can add new methods and private instance variables.
III `Teacher` can override existing private methods of `Person`.

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) II and III only

13. Consider the following hierarchy of classes:



A program is written to print data about various birds:

```
public class BirdStuff
{
    public static void printName(Bird b)
    { /* implementation not shown */ }

    public static void printBirdCall(Parrot p)
    { /* implementation not shown */ }

    //several more Bird methods

    public static void main(String[] args)
    {
        Bird bird1 = new Bird();
        Bird bird2 = new Parrot();
        Parrot parrot1 = new Parrot();
        Parrot parrot2 = new Parakeet();
        /* more code */
    }
}
```

Assuming that none of the given classes is abstract and all have default constructors, which of the following segments of `/* more code */` will *not* cause an error?

(A) `printName(parrot2);`
`printBirdCall((Parrot) bird2);`

(B) `printName((Parrot) bird1);`
`printBirdCall(bird2);`

(C) `printName(bird2);`
`printBirdCall(bird2);`

(D) `printName((Parakeet) parrot1);`
`printBirdCall(parrot2);`

(E) `printName((Owl) parrot2);`
`printBirdCall((Parakeet) parrot2);`

Refer to the classes below for Questions 14 and 15.

```
public class ClassA
{
    //default constructor not shown ...

    public void method1()
    { /* implementation of method1 */ }
}

public class ClassB extends ClassA
{
    //default constructor not shown ...

    public void method1()
    { /* different implementation from method1 in ClassA*/ }


    public void method2()
    { /* implementation of method2 */ }
}
```

14. The `method1` method in `ClassB` is an example of
    (A) method overloading.
    (B) method overriding.
    (C) polymorphism.
    (D) information hiding.
    (E) procedural abstraction.

15. Consider the following declarations in a client class.

```
ClassA ob1 = new ClassA();
ClassA ob2 = new ClassB();
```

Which of the following method calls will cause an error?

I `ob1.method2();`

II `ob2.method2();`

III `((ClassB) ob1).method2();`

(A) I only
(B) II only
(C) III only
(D) I and III only
(E) I, II, and III

Use the declarations below for Questions 16–18.

```
public abstract class Solid
{
    private String name;

    //constructor
    public Solid(String solidName)
    { name = solidName; }

    public String getName()
    { return name; }

    public abstract double volume();
}

public class Sphere extends Solid
{
    private double radius;

    //constructor
    public Sphere(String sphereName, double sphereRadius)
    {
        super(sphereName);
        radius = sphereRadius;
    }

    public double volume()
    { return (4.0/3.0) * Math.PI * radius * radius * radius; }
}

public class RectangularPrism extends Solid
{
    private double length;
    private double width;
    private double height;

    //constructor
    public RectangularPrism(String prismName, double l, double w,
            double h)
    {
        super(prismName);
        length = l;
        width = w;
        height = h;
    }

    public double volume()
    { return length * width * height; }
}
```

16. A program that tests these classes has the following declarations and assignments:

```
Solid s1, s2, s3, s4;
s1 = new Solid("blob");
s2 = new Sphere("sphere", 3.8);
s3 = new RectangularPrism("box", 2, 4, 6.5);
s4 = null;
```

How many of the above lines of code are incorrect?

(A) 0
(B) 1
(C) 2
(D) 3
(E) 4

17. Which is *false*?

(A) If a program has several objects declared as type `Solid`, the decision about which `volume` method to call will be resolved at run time.
(B) If the `Solid` class were modified to provide a default implementation for the `volume` method, it would no longer need to be an abstract class.
(C) If the `Sphere` and `RectangularPrism` classes failed to provide an implementation for the `volume` method, they would need to be declared as abstract classes.
(D) The fact that there is no reasonable default implementation for the `volume` method in the `Solid` class suggests that it should be an abstract method.
(E) Since `Solid` is abstract and its subclasses are nonabstract, polymorphism no longer applies when these classes are used in a program.

18. Here is a program that prints the volume of a solid:

```
public class SolidMain
{
    /** Output volume of Solid s. */
    public static void printVolume(Solid s)
    {
        System.out.println("Volume = " + s.volume() +
                " cubic units");
    }

    public static void main(String[] args)
    {
        Solid sol;
        Solid sph = new Sphere("sphere", 4);
        Solid rec = new RectangularPrism("box", 3, 6, 9);
        int flipCoin = (int) (Math.random() * 2);   //0 or 1
        if (flipCoin == 0)
            sol = sph;
        else
            sol = rec;
        printVolume(sol);
    }
}
```

Which is a true statement about this program?

(A) It will output the volume of the sphere or box, as intended.
(B) It will output the volume of the default `Solid s`, which is neither a sphere nor a box.
(C) A `ClassCastException` will be thrown.
(D) A compile-time error will occur because there is no implementation code for `volume` in the `Solid` class.
(E) A run-time error will occur because of parameter type mismatch in the method call `printVolume(sol)`.