1 Overloading vs. overriding

Consider the following declarations:

```java
class Animal {
    void eat(Plant p) { System.out.println("Mmm, veges!"); }
    void eat(Animal a) { System.out.println("Mmm, meat!"); }
}
class Marsupial extends Animal {
    void eat(Animal a) { System.out.println("Meh."); }
}
class Koala extends Marsupial {
    void eat(Eucalyptus e) { System.out.println("Nom nom nom"); }
    void eat(Plant p) { System.out.println("Uh-oh"); }
    void eat(Animal a) { System.out.println("No thanks, I’m a vegan."); }
}
class Cow extends Animal {
}
class Plant {
}
class Eucalyptus extends Plant {
}

Koala koala = new Koala();
Marsupial mars = koala;
Animal steak = new Cow();
Eucalyptus euc = new Eucalyptus();
Plant plant = euc;
```

What is printed by executing the following Java code?

1. mars.eat(plant);
2. mars.eat(euc);
3. koala.eat(plant);
4. koala.eat(euc);
5. koala.eat(mars);
6. koala.eat(steak);
2 Generics

Consider the following generic classes in Java:

class A { int a; }
class B extends A { int b; }
class C<T> { ... }

And consider the following code that uses these classes:

C<B> ob = new C<B>();
C<A> oa = ob;
...

Java will reject this program because it won’t type-check the assignment to oa. If you think it is unsafe to run this program, fill in the missing code (indicated by “...”) to show how this can lead to a run-time type error.

3 Inheritance

In several OO languages, including C++ and Eiffel, a subclass can hide members of the superclass. In C++, for example, we can declare a base class to be public, protected, or private:

class B : public A {
    // public members of A are public members of B
    // protected members of A are protected members of B
}
class C : protected A {
    // public and protected members of A are protected members of C
}
class D : private A {
    // public and protected members of A are private members of D
}

In all cases, private members of A are inaccessible to methods of B, C, or D.

Consider the impact of protected and private superclasses on dynamic method binding. Under what circumstances can a reference to an object of class B, C, or D be assigned into a variable of type A*. Should B*, C*, or D* be considered subtypes of A*?
4 Object implementation

Consider the Java program below:

```java
interface Pingable {
    public void ping();
}

class Counter implements Pingable {
    int count = 0;

    public void ping() {
        ++count;
    }

    public int val() {
        return count;
    }
}

public class Ping {
    public static void main(String[] args) {
        Counter c = new Counter(); // (a)
        c.ping();
        c.ping();
        int v = c.val();
        System.out.println(v);
    }
}
```

Assume this is compiled onto a machine with 32-bit (i.e., 4-byte) addresses.

1. Draw a picture of the layout in memory of the `Counter` object created in `main` and labeled (a). Include details such as the dispatch vector.

2. Give assembly-level pseudocode for the call to `c.val` in `main`. Assume the address of `c` is in register `r1` immediately before the call and the same register is used to pass the hidden `this` parameter to the method. Ignore details like saving and restoring registers or where to put the return value. Assume there are instructions for loading and storing from an address with a constant offset, for arithmetic, for calling procedures.

3. Give assembly-level pseudocode for the body of the method `Counter.ping` (again ignoring register saves and restores). Assume `this` is in register `r1`. 