Recitation 1

Inheritance, Static Members

---

1. Consider the following class definition:

```
public class A {
  public A () {}
  public A (int a, int b) {}
}

public class B extends A {
    public B (int r) {}
    public B (int r, int w) {
      super (r, w);
    }
}
```

Which of the following are legitimate calls to construct instances of the B class? For those that are not, explain why.

a. `B c = new B();`
b. `A s = new B(1);`
c. `B c = new B(1, 9);`
d. `A t = new B(1, 9, 4);`
e. `B t = (new B(1)).new B(1);`
f. `B b = new A(1, 2);`

---

2. Inheritance/Dynamic Binding

a. Will the following code compile? If not, where exactly will it fail to compile?

```
public class A {
  public int x;
  public A(int x) {
    this.x = x;
  }
}

public class B extends A {
  public int y;
  public B(int y) {
    this.y = y;
  }
}
```

b. Given:

```
public class B {
  public int x;
  public String toString() {
    return x + "";
  }
}
```

```
public class E extends B {
  public int y=3;
  public String toString() {
    return (x + y) + "";
  }
}
```

What is the output of the following code segment:

```
B b = new E();
b.x = 5;
System.out.println(b);
```

c. Given:

```
public class B {
  private int x;
  public int getX() {
    return x;
  }
  public String toString() {
    return x + "";
  }
}

public class E extends B {
  public int y=3;
  public String toString() {
    return getX() + y + "";
  }
}
```

What is the output of the following code segment, which is in a different class than B or E:

```
B b = new E();
System.out.println(b)
```

d. Given:

```
public class V {
  public static int stuff() {
    return 1;
  }
}

public class W extends V {
  public static int stuff() {
    return 2;
  }
}
```

What is the output of the following code segment, which is in a different class than W or V:

```
V v = new W();
System.out.println(v.stuff());
```

e. Given:

```
public class G {
  public int g;
}

public class H extends G {
  public int h;
  public boolean equals(Object o) {
    if (o == null || !(o instanceof H)) {
      return false;
    }
    return g == ((H)o).g;
  }
}
```

What is the output of the following code segment, which is in a different class than H or G:

```
G ag = new H(); ag.g = 15;
G bg = new G(); bg.g = 15;
if (ag.equals(bg)) {
  System.out.println(10);
} else {
  System.out.println(20);
}
```

f. Given:

```
public class B {
 public int x;
 public String toString() {
   return x + "";
 }
}

public class E extends B {
 public int y=3;
 public String toString() {
   return (x + y) + "";
 }
}
```

What is the output of the following code segment, which is in a different class than B or E:

```
B b = new E();
System.out.println(b.y)
```

---

3. What is the output of this code? Why?

```
class GrandParent { }
class Parent extends GrandParent{ }
class Child extends Parent { }

class Foo {
  public void bar(GrandParent p) {
    System.out.println("called with type GrandParent");
```

```
  }
  public void bar(Parent p) {
    System.out.println("called with type Parent");
  }
}

public class Test {
  public static void main(String[] args) {
    new Foo().bar(new Child());
  }
}
```

---

4. In this exercise we will try to see how static and final methods work with inheritance. (A final method is one that cannot be overridden in a subclass.) Consider the two classes defined below, Parent and Child:

```
public class Parent {
  /*
   public static final void printClassName() {
     System.out.println("I am in class Parent, static invocation.");
   }
  */

   public final void printName() {
     System.out.println("I am in class Parent, dynamic invocation.");
   }
}

public class Child extends Parent {
  /*
   public static void printClassName() {
     System.out.println("I am in class Child, static invocation.");
   }
  */

   public void printName() {
     System.out.println("I am in class Child, dynamic invocation.");
   }
}
```

   A. Compile both the classes. What do you see?
   B. Now comment out the printName() method in the Child class and uncomment the printClassName() method in both classes. Compile both classes. What do you see? Is it different from part (A)? Why?
   C. Uncomment the Child.printName() method and remove the final modifier from Parent.printName() and Parent.printClassName(). Recompile both classes.
   D. Compile and run the following class:

```
class App {
  public static void main(String[] args) {
    Parent p = new Child();
    p.printName();     // WHAT IS PRINTED?
    p.printClassName(); // WHAT IS PRINTED?
  }
}
```

Explain the difference in *which* methods are invoked in these two method calls.

---

5. Here's a Widget class:

```
public class Widget {
  float mass;

  private static float MAX_MASS = 20;

  public static final float G = 9.81f;

  public Widget(float mass) {
    if (mass > MAX_MASS) {
      throw new IllegalArgumentException();
    }
    this.mass = mass;
  }

  public static float getMaxMass() {
    return MAX_MASS;
  }

  public float getWeight() {
    return mass*G;
  }
}
```

Now suppose there is a certain set of widgets that are "heavy", so their maximum mass is 40 instead of the usual 20. Write a class called HeavyWidget, as a subclass of Widget. Do you encounter any implementation issues when you do this? Can you get around these issues? If so, show how. If not, explain why.