**David Yang**

**Jacob Hanna**

**Lab 4**

**Part 1**

**a)** we can assign ya to xa, because it is upcasting. So, we can assign xa = ya; But assigning xa to ya is a downcasting, so it requires explicit casting. ya = (Y[])xa;

**b)** Yes we can do it by explicit casting, if we have a back-up copy of array of Y's to assign back the array of Y's to ya.

**c)** Yes, we can do it. If we have a back-up copy of array of X's. Here the casting will occur implicitly. No explicit casting is required.

Code: public class Test {  
  
public static void main(String[] args) {  
// write your code here  
  
X x = new X();  
Y y = new Y();  
  
X[] xa = new X[10];  
Y[] ya = new Y[10];  
  
/\*  
x = y;  
  
y =(Y) x;  
  
xa = ya;  
  
ya = (Y[]) xa;  
\*/  
  
  
}  
}

**Part 2**

**a)** Yes, it will be compiled if the subclass implements the interface method if it has the same return type, same parameters, and same name. Parameter names need not be the same.

**b)** If the method declaration has a different return type then it will not compile. It is not considered as an implemented method. And it will compile only after the interface method is implemented.

**c)** In this case, also compilation fails because of the difference in parameter type.

**d)** It will be compiled. If there is a difference in parameter names, there will be no problems.

**Part 3**

1. Yes, it will compile. It doesn’t matter.
2. Java compiles, and it doesn’t make a difference.
3. Here you go: public class random {  
    public static void main(String[] args){  
    rando a= new hello();  
    System.*out*.println(a.*a*);  
    }  
   }  
   class rando{  
    public static final int *a*=0;  
   }  
   interface blabla{  
    public static final int *a*=2;  
   }  
   class hello extends rando implements blabla{  
     
   }

**Part 4**

**a)** Java obviously calls the subclass variable’s method.

public class random {  
 public static void main(String[] args){  
 rando a= new hello();  
 ((rando)a).printa();  
  
  
 }  
}  
class rando{  
 public void printa(){System.*out*.println("a");}  
}  
class hello extends rando {  
  
  
 @Override  
 public void printa(){  
 System.*out*.println("b");  
 }  
  
  
}

**b)** To downcast, you cannot go and just try to cast it, it’s going to throw a ClassCast Exception

rando a= new rando();  
((hello)a).printa();

**c)** Yes, if we do it like this. Or we can overload printa, but I prefer this way. If we want to make variable a’s parent class rando, we can also do that, but we need to downcast.

public class random {  
 public static void main(String[] args){  
 hello a= new hello();  
 a.callparent();  
  
  
 }  
}  
class rando{  
 public void printa(){System.*out*.println("a");}  
}  
class hello extends rando {  
  
 public void callparent(){  
 super.printa();  
 }  
  
 public void printa(){  
 System.*out*.println("b");  
 }  
  
  
}