Aufgabe 1

a)
//1
Integer j = i --> Autoboxing (automatischer Aufruf von Integer.valueOf(i) )
Integer k = new Integer(i) --> neues Objekt der Wrapperklasse Integer wird erstellt durch den Konstruktor-Aufruf von *Integer*
//2
Die Ausgabe ist *true,* da „i“ vom primitiven Datentyp *int* ist wird j automatisch unboxed und die beiden Werte werden verglichen.
//3
Hier ist die Ausgabe *false* . „j“ ist ein autogeboxtes Integer und „k“ ein Integer mit Konstruktoraufruf. Daher haben wir einmal ein Integer Objekt (k) und einmal ein geboxtes Integer mit der Methode Integer.valueOf. Beides unterschiedliche Objekte, daher keine Identiätengleichheit.
//4
Da es für Integer keine Operation >= bzw. <= werden k und j automatisch unboxed und dann werden die Werte verglichen.
Da k>=j && k<=j dem mathematischen k=j entspricht, werden somit die Werte von k und j verglichen.
Also ist die Ausgabe true, da beide den Wert 42 haben.
//5
In Zeile 22 wird k die Objektreferenz von j zugewiesen.
In Zeile 23 wird zuerst Integer j unboxt, mit 5 multipliziert und anschließend wieder in ein Integer geboxt.
In Zeile 24 wird Integer j unboxt
//6
Die erste Ausgabe ist false: Es werden zwei Integer auf Gleichheit mit "==" überprüft was nur true zurückgibt wenn beide auf das selbe Objekt referenzieren.
Die zweite Ausgabe ist *true : Es werden zwar wieder zwei Integer mit "==" verglichen diesesmal referieren sie aber auf das selbe Objekt, da eine Integer Liste für die Objekte zwischen -128 und +12 bereits bei Programmstart eingerichtet wurde und alle weiteren Objekte zwischen -128 und +127auf diese entsprechend referieren.*

b)
In Zeile 1 werden die einzelnen *chars* in die Wrapperklasse Charakter geboxt und ein Array derselben angelegt. In Zeile zwei soll nun diese wieder ein char-Array umgewandelt werden. Das funktioniert in Java so nicht. Es kommt hier also zu einem Fehler, da Arrays nicht automatisch konvertiert werden können.
In Zeile 4 findet hier ein Autoboxing statt.
In Zeile 5 soll nun das Integer in ein String konvertiert werden, was einen Fehler auftreten lässt. Weder ein *int* noch ein *Integer* lassen sich in einen String umwandeln. Hier müsste man *parsen.*

c)
Man schreibt hier die generische Klasse so, dass Objekte dieser Klasse serialisierbar und vergleichbar sind.

d) Covarianz bedeutet, die Typhierachie entspricht der Vererbungsrichtung.
Nun wollen wir auf die Objektvariable Name zugreifen, die in der Klasse Tier deklariert ist. In der Methode getName der Klasse Tier sind als Übergabewerte beispielsweise nur Tiere und von Tiere abgeleitete Klassen erlaubt. Ruft man dieselbe Methode in der von Tier

abgeleiteten Klasse Elefant auf, so sind dort nur Übergabeparametere vom Typ Elefant zulässig. Deshalb kann die Methode getName der Klasse Tier zwar aus der Methode getName des Typs Elefant lesen, allerding nicht hineinschreiben, da sonst eventuell eine Typ einer anderen Ableitung der Klasse Tier übergeben wird(zum Beispiel Maus), der in der Klasse Elefant überhaupt nicht zulässig ist.

e)Kontravarianz bezieht sich auf einen Aspekt, entgegengesetzt zur Vererbungsrichtung.

f)Invarianz bedeutet, dass die Unterklasse die gleichen Typen akzeptiert wie die Oberklasse. Invarianz erlaubt sowohl lesenden als auch schreiben Zugriff.

g) Durch Wildcardtypen wird die Flexibilität erhöht. Syntax: C<?>
Wenn wir zum Beispiel einen Typ C erstellen wollen, bei dem uns egal welche primitive Datentypen er beiinhaltet schreiben wir C<?>.

h)Eine Wildcard darf nicht zur Instanzierung eines Objekts verwendet werden, da Wildcards abstrakt sind.