(1) The given code snippet is partially correct. The array size is 1. However, it tries to assign two String objects to array.

```
// Fixed solution
String[] strings = new String[2];
strings[0] = ‘‘Hello’’;
strings[1] = ‘‘World’’;
```

(2) The "++" operator cannot be applied to String objects.

(3) The given code snippet does not provide explicit casting when it assigns a variable with a larger type to a variable with a smaller type. When a variable is assigned to a variable with smaller type, it requires narrowing casting mannually since it might experience lost in information or overflow (underflow).

```
long longVar = 42L;
short shortVar = 42;
shortVar = (short) longVar;
```

(4) ArrayList does not support primitive types since ArrayList is a class that supports generic types, and generic types require reference types as type parameters. An solution is to make use of wrapper classes.

```
// Fixed Solution
ArrayList<Integer> ints = new ArrayList<Integer>();
ints.add(3);
```

(5) One example is the use of wrapper classes for primitive types. The use of wrapper classes comes with overhead at the cost of memory requirement as well as the process of boxing. Another example is the location of throwing exception. The location of throwing exception depends on the situation, and also decides who should have the responsibility to handle and resolve that exception. This utlimaitely relies on the agreement and decision between the developers and the clients, which usually takes a consideration and discussion to finalize.

6 (a) Apple extends Food logically make sense. Apple can be consumed by most of the animals to obtain nutrition. An alternative way is the set Apple class to extends Fruit class, which extends Food class.

6 (b) Ferrari extends RaceCar logically make sense. Most of the car models in Ferrari can be labelled as race car because of the outstanding performance

6 (c) Dogs extends Animal logically make sense (self explanatory).

6 (d) JustinBieber extends Artist does not make sense. Justin Biber is a singer with concrete characteristics, thus logically JustinBiber should be an instance (object) of Artist.

6 (e) Oven extends Kitchen does not make sense. Oven is a tool / cookware in the kitchen, which indicates that a "is-a" relationship (Kitchen has an oven). Also, kitchen describe a specific environment and space, which does not suit to the characteristics of an oven.

7 (a) This is not allowed because class Animal is an abstract class, and cannot be instaniated.

7 (b) This is allowed because of polymorphism. Class Dog inherits Class Canine, which inherits Class Animal. Thus, an object of Class Dog is also an object of Class Animal. Logically, it also make sense that dog is one type of an animal.

7 (c) This is allowed. The code snippet intialized an Canine object that matches the type of **canine1**.

7 (d) This is not allowed. An object of class Dog can be an object of class Canine. However, the logic does not apply in the reverse. An object of class Canine might be an object of class Dog but it can be also object of other class that inherits class Canine but not class Dog.

7 (e) This is allowed. All classes in java inherits class Object.

7 (f) This is allowed. eat() in class Canine is called.

7 (g) This is allowed. makeSound() in class Animal is called.

7 (h) This is allowed. makeSound() in class Dog is called.

7 (i) This is not allowed. The reference type is Object in compile time. eat() is specifically created in class Dog. It does not exist in class Object when checking whether the reference type has eat() method.

7 (j) This is not allowed. run() method in class Animal is abstract. It only contains the method signitrue without the body / implementation.