Practice Problems for the Final

Questions

1. Consider a petting zoo class which is implemented with parametric polymorphism in some unknown language:

```
// petting_zoo.src
class PettingZoo<T> {
public:
   void add(T animal);
   void pet_all_animals() {
      for animal in animals:
         animal.pet();
   }
   …
private:
   Array<T> animals = new Array<T>;  // allocate array for animals
};
```

Here's how it'd be used:

```
// main.src
PettingZoo<Dog> dog_petting_zoo = new PettingZoo<Dog>;
Dog shep = new Dog();
dog_petting_zoo.add(shep);
dog_petting_zoo.pet_all_animals();
```

a. Let's assume this class is ONLY used with Dogs. Is the above code using generics or templates? How can you tell?

b. Let's assume that the code is using the alternate scheme than the one you identified in part a (e.g., if you guessed Generic in part a, then the alternate scheme would be Templating, and visa-versa). What would you have to change about the PettingZoo class to allow it to properly let you pet animals in the petting zoo? Why would you have to do this?

c. Assume you created petting zoos for dogs, giraffes and monkeys in a single program, and compiled the program to an executable file. Would the executable file be smaller, the same size, or larger if the language were using Generics rather than Templates? Why?

2. Consider the following program in an unknown language:

```
def foo(x: int):
   x = 3.5  // Line A
   print(x)

def main():
  foo(10)
```

a. Assuming this language is dynamically typed, what would be the type of the value pointed to by x after line A?
b. Assuming the language is dynamically typed, what is the type of variable x after line A?
c. Assuming the language is statically typed, what would be the type of the value pointed to by x after line A?
d. Assuming the language is statically typed, what is the type of variable x after line A?
e. Assuming the language is statically typed, what must be happening on line A to perform the assignment.

3. We have a Washable interface and a Driveable interface, and define a Car class which implements both interfaces - it can be washed and driven. Which of the following are true:

a. Car is a supertype of Washable and Driveable
b. Car is a subtype of Washable, and a subtype of Driveable
c. Washable and Driveable are supertypes of a Car
d. Washable is a reference type
e. Car is a reference type
f. If a Car indicates that it supports both Washable and Driveable interfaces, but ONLY implements the Washable methods, but not the Driveable methods, Car would be a concrete type that could be used to define objects

Solutions

1.

a. Is the above code using generics or templates? How can you tell?

The code must be using templates, because generics do not allow specific member functions like pet() to be called within the generic. A generic must only rely upon the lowest-common denominator functionality across all objects. So with a generic, the pet() method call would be prohibited. A template-based generic would allow this, however.

b. Let's assume that the code is using the alternate scheme than the one you identified in part a (e.g., if you guessed Generic in part a, then the alternate scheme would be Templating, and visa-versa). What would you have to change about the PettingZoo class to allow it to properly work? Why would you have to do this?

You'd have to define a Pettable interface which contains at least a pet() function, and use that to bound your generic, e.g.:

```
interface Pettable {
  void pet();
}

class PettingZoo<T: Pettable> {
public:
   void add(T animal);
   void pet_all_animals() {
      for animal in animals:
         animal.pet();
   }
   …
private:
   Array<T> animals = new Array<T>;  // allocate array for animals
};
```

The specific syntax doesn't matter, since we didn't tell you the syntax of the language. But you need to know that you'd have to bound the generic with a pettable interface.

c. Assuming you created petting zoos for dogs, giraffes and monkeys in a single program, and compiled the program to an executable file. Would the executable file be smaller, the same size, or larger if the language were using Generics rather than Templates? Why?

If the language used generics, the program would be smaller. When using templates, the compiler generates a separate copy of the class for EVERY single type its used with, meaning that it would generate and compile code for PettingZoo<Dog>, PettingZoo<Giraffe>, and PettingZoo<Monkey>, making the code much larger.

2

a. Assuming this language is dynamically typed, what would be the type of the value pointed to by x after line A?

The type of the value would be float or double, since the variable x refers to a floating point number after the assignment.

b. Assuming the language is dynamically typed, what is the type of variable x after line A?

   Variables do not have type in dynamically typed languages. The variable is just a name that is bound to a value, and the value has a type.

c. Assuming the language is statically typed, what would be the type of the value pointed to by x after line A?

   The type of the value would be int, since the variable x was defined as an integer in the formal parameter list, and assigning it to a float wouldn't change the original type.

d. Assuming the language is statically typed, what is the type of variable x after line A?

   The type of the variable would be int, since the variable x was defined as an integer in the formal parameter list.

e. Assuming the language is statically typed, what must be happening on line A to perform the assignment.

   There must be a coercion (an implicit conversion) between float to int during the assignment.

3.

a. Car is a supertype of Washable and Driveable: False
b. Car is a subtype of Washable, and a subtype of Driveable: True
c. Washable and Driveable are supertypes of a Car: True
d. Washable is a reference type: True
e. Car is a reference type: False
f. If a Car indicates that it supports both Washable and Driveable interfaces, but ONLY implements the Washable methods, but not the Driveable methods, Car would be a concrete type that could be used to define objects: False - Car would be an abstract class, requiring subclassing and implementation if the unimplemented methods to create a concrete class that can be used to instantiate objects.