Questions

11/18 points

What is programming paradigm?

- It is a programming style based on specific programming language pros and cons.
- A good design pattern.
- Step-by-step instructions to follow.
- A set of concepts, instruments, principles that define the fundamentals of programming style. ✔

Answer

Correct:

That's right! it is a method to solve a problem using tools and techniques that are availab
us following some approach.

Which paradigm dictates WHAT should be done but not HOW it should be done?

- Imperative

- Object Oriented
- Declarative ✔
- Functional

Answer
Correct:
You are absolutely correct! In declarative programming, the programmer instructs the computer on what is to be computed. You do not know how it works, but you know what it does.

Which paradigm makes use of methods and properties to encapsulate data in classes?

- Imperative
- Object Oriented ✔
- Declarative
- Functional

Answer
Correct:
Keep it up! Object-oriented programming represents a specific way of organising code, and it can be defined as 'an entity that encapsulates both data and behaviour'. This means that the data and all the methods of a system belong to one or more classes.

Which paradigm involves using immutable data?

- Functional ✔
- Imperative
-

Declarative

Object Oriented

Answer
Correct:
Correct! Functional programming pushes us towards the idea that it would be nice to use immutable data. Since immutable data is faster, because we put them into memory once, and they do not change.

Why do we need UML for?

for describing system components and their interrelationships

for describing set of rules for developing our application

for describing design for storing and transporting data

for describing relations between entities in our application ✖

Answer
Incorrect:
That's not quite right! Describing relations is only one of techniques that UML provides us.

Once classes and attributes have been identified and placed into a diagram, the next stage is to add?

Operations

Multiplicities

Actors

Associations ✔

Answer
Correct:
That’s right! Usually, the system is not limited to just one class, there are tens, hundreds of classes, or even more. Naturally, all these classes somehow interact with each other, somehow communicate, send messages to each other, call each other's methods, send events, and so on. So the next step after creating those classes should be the visual representation of the relationship between these classes.

What type of relationship should be used if base class acquires all the properties and behaviors of its successor?

- Association
- Inheritance ✔
- Composition
- Aggregation

Answer
Correct:
Keep it up! Inheritance is a more precise type of relationship (IS A Relationship), which says that everything that is true for the base class is true for its successor.

What is Abstraction?

- The provision of a single interface to entities of different types or the use of a singl symbol to represent multiple different types.
- The act of representing essential features without including the background details or explanations.
- The tool that helps to hide unimportant implementation details out of sight.
- The mechanism of basing an object or class upon another object.

Answer
Correct:
Correct! Abstraction in object-oriented programming means the highlighting of some significant parts, meaningful information from a component, no matter whether it is a class or an architectural layer in the system, or a logical unit of our system.

What is Encapsulation?

- Concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse.
- Concept that specifies the shared communications protocols and interface methods used by hosts in a communications network.
- The tool that helps to hide unimportant implementation details out of sight. ✖
- Selecting data from a larger pool to show only the relevant details to the object.

Answer
Incorrect:
That's not quite right. While it seems from the name, hiding implementation details is only consequence of bundling of data and methods operating on this data into one unit.

What is Polymorphism?

- The mechanism of basing an object or class upon another object.
- The provision of a single interface to entities of different types or the use of a single symbol to represent multiple different types.
- Concept that specifies the shared communications protocols and interface methods used by hosts in a communications network.
- The provision of a different interfaces to a single entity to represent multiple different types. ✖

Answer
Incorrect:
That’s not quite right. Polymorphism is a concept when objects of different types are sharing the same interface.

What is Inheritance?

- Provision of a single interface to entities of different types or the use of a single symbol to represent multiple different types.
- Mechanism of basing an object or class upon another object. ✔
- Concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse.
- The act of representing essential features without including the background details or explanations.

Answer
Correct:
In most class-based object-oriented languages, an object created through inheritance (a "child object") acquires all the properties and behaviors of the parent object.

Which inheritance model is used in JavaScript?

- Prototypal model ✔
- Object model
- Functional model
- Class model

Answer

Correct:
Correct! JS uses prototypes to set the connection between objects which allows to build the inheritance structure.

When we should use an inheritance?

- ◯ When we can share some common behavior among few specific subclasses. ✔
- ◯ When there complex public interface which can be moved to the abstract class.
- ◯ When the class becomes hard to maintain.
- ◯ When we can split the class on two parts to decrease its size.

Answer
Correct:
Correct! This is the most valuable reason to use the inheritance as it almost never makes sense to create a single subclass, but if we have some behavior which can be shared among few subclasses it is a sign to use the inheritance.

Choose the statements which are correct for abstract class:

- ☐ can store both abstract and concrete methods
- ☐ should have a single subclass
- ☐ can be instantiated (class instance can be created)
- ☐ all the abstract methods must be implemented in every non-abstract subclass
- ☐ stores the behaviour which is common to all subclasses
- ☐ should always implement an interface

✔

Answer
Correct:
Absolutely true, while abstract methods must be implemented to use, concrete methods can be used without overriding them.
You always need to implement all the abstract methods and there should be no unused behavior in any subclass, if such thing happens this may be a sign of incorrect inheritance structure.
That's true, subclasses inherited an abstract class should fully use its functionality, otherwise you need to review the inheritance structure.

Template method is a:

- ○ method which is defined in an abstract class and cannot be redefined in a subclass
- ◯ method which is called each time when the class instance is created ✖
- ○ technique of describing the basic algorithm in a superclass and redefining its parts in a subclass
- ○ method which can be shared among different classes

Answer
Incorrect:
While it is true that template method is called each time when instance is created the main idea is to give the possibility to override this behaviour in subclasses, but not just call it each time.

When should class implement an interface?

- ☐ When external environment depends on it.
- ☐ When class implements Strategy pattern.
- ☐ When this class is abstract.
- ☐ When class is an adapter.
- ☐

- [ ] In any case when it is possible.
- [ ] When class implements role interface.
- [ ] when class needs to communicate with object of another level.

✖

Answer
Incorrect:
It is a good reason to implement an interface for a class, this will provide a testability to its users.
As result of Interface Segregation Principle class needs to implement the role interface, like IClonable, IComparable etc.
When class needs to communicate with object of another level it may depend on but not necessary needs to implement the interface.

Choose the statements which are correct for any public interface:

- [ ] handle implementation details
- [ ] always referenced in the tests
- [ ] is safe to depend on
- [ ] reveal its primary responsibility
- [ ] may change on any reason
- [ ] is expected to be invoked by others
- [ ] can be implemented only by abstract classes

✖

Answer
Incorrect:
Unit tests is used to describe the smallest parts of the functionality and use of interfaces will only help to identify when something was broken during the refactoring.

One of the main principles of public interface is not to change on any reason, because this will change in all the places where it is used, that's why it is generally safe to depend on them. As one of the primary purposes of any interface is to describe the communication between objects, it is expected to be invoked by different parts of the application.

Which of the following are bad design smells?

- [ ] Simplified Methods Calls
- [x] Viscosity
- [ ] Dealing with Generalization
- [ ] Immobility
- [x] Fragility
- [ ] Organized Data
- [x] Rigidity
- [ ] Composing Methods
- [x] Needless Complexity

✖

Answer

Incorrect:

The system is viscous if basic operations are difficult or take too long to complete.

That's right! Fragility indicates that a system is fragile if a change in some part breaks something in another part.

Navigation

change will entail high costs.

That's right! It indicates that the system is unnecessarily complicated or prematurely optimized, has too much code that is not currently used.

Submit

✱ Partially correct (11/18 points)

Complete «Building Blocks of OOP. Part 1»