1. Which of the following are explicit activities in the Model Exploration Whirlpool?

A. Code Probe
B. Production Release
C. Model
D. Demo
E. Coffee Break
F. Scenario
G. Iteration Planning
H. Harvest and Document

2. Which of the following are true statements about the Model Exploration Whirlpool?

A. The team must complete the Whirlpool at the start of a project, as part of the Design Phase
B. The Whirlpool shifts modeling from "push" to "pull"
C. The team can only jump into the Whirlpool once they have a Reference Scenario
D. Only developers are involved in the Whirlpool
E. The Whirlpool never involves writing actual code – why waste time doing *that*?
F. Brainstorming sessions are an integral part of the Whirlpool
G. The Whirlpool cultivates the creative collaboration of the technical team and their business counterparts
H. Selecting the right scenarios and describing them properly is critical to successful modeling

3. Which of the following are indicators that it could be time for modeling?

A. Communications with stakeholders deteriorate
B. That phase has been reached in the software development lifecycle (SDLC)
C. Solutions seem more complex than the problems
D. Velocity slows (completed work becomes a burden)
E. Management says so
F. The software will be released to production tomorrow
G. The team is moving into a new domain area for the first time and needs a minimal shared view to get started

4. What are important attributes of reference scenarios?

A. Written up as a User Story, and prioritized as such in the Product Backlog
B. Describe an important (complex/risky) business problem the software needs to solve
C. Provide shared understanding of business domain between domain experts and technical team
D. Documented in standard Use Case format
E. Harvested and documented for later reference in future modeling sessions
F. Diagrammed in UML, and only UML
G. Concrete, realistic and specific

5. In Domain-Driven Design (DDD), what do we mean by "domain"?

A. A layer in the architecture where the business logic is kept
B. A sphere of knowledge, influence, or activity (the subject area to which the user applies a program)
C. The same thing as a "bounded context"
D. A model
E. The first word in the name of the book title

6. When logic is complex, look for implicit concepts. Where are likely places to find clues for implicit concepts?

A. In the way spoken statements are phrased.
B. Ambiguous and confusing unit tests
C. In the awkwardness of particular parts of the design
D. Astrological charts
E. Within code obscured by many implementation details or complicated compensation logic

7. When collaborating with domain experts, what are important things to keep in mind?

A. Cultivate a strong relationship between the domain experts and development team
B. Try to get to the requirements as quickly as possible and ignore everything else domain experts say as it's probably boring and irrelevant
C. Restrict the conversation to only what the software has to do
D. Focus on the ubiquitous language
E. Once you can get the right answer, you are done. Yes!
F. Listen for clues
G. You must literally *become* a domain expert if you want to succeed with DDD
H. Be open to shifts in assumptions that destroy your model
I. Keep definitions as vague and abstract as possible
J. Ask for recommended reading
K. Build a relationship of mutual respect
L. Avoid walkthroughs, they will only distract you from modeling
M. Learn enough to have an intelligent conversation

8. According to definitions given in class, which of the following statements about models are true?

A. A model is a system of abstractions representing selected aspects of the domain
B. Usefulness specific to particular scenarios is how we evaluate models
C. UML diagrams are the only valid way to describe a model
D. Models must be as realistic (i.e. close to the real world) as possible
E. Models are important because the critical complexity of most software projects is in understanding the domain itself
F. They must be anemic to be useful
G. Models can be used to solve problems related to the domain
H. A model is a distilled form of domain knowledge, assumptions, rules and choices
I. There is only ever one model of the domain

9. What of the following could be done to improve the following code from a DDD perspective, as we did in the hands-on coding exercises in class?

```
 public void reroute(Cargo cargo, String reroutePoint)
{
    Itinerary itinerary = cargo.getItinerary();
     List<Leg> legs = new ArrayList<Leg>();
    legs.addAll(cargo.getItinerary().getLegs());

    boolean pastReroutePoint = false;

    for(Leg leg:legs)
    {
        if (leg.getStart().equals(reroutePoint))
               pastReroutePoint = true;
        if (pastReroutePoint) cargo.getItinerary().remove(leg);
    }
    Cargo tempCargo = new Cargo();
    tempCargo.setOrigin (reroutePoint);
    tempCargo.setDestination(cargo.getDestination());
    route(tempCargo);
    for (Leg leg :tempCargo.getItinerary().getLegs())
    {
        cargo.getItinerary().add(leg);
    }
}
```

A. Move responsibilities for performing complex operations to appropriate objects (eg. such as Itinerary)

B. Return a new Cargo entity from the reroute() method

C. Look for concepts that are implicit in the code, and try to express them concisely and explicitly

D. Rewrite it in C#, a *real* programming language

E. Define preconditions and postconditions for the reroute() method

F. Make the new destination explicit by passing it in as a parameter

G. Don't use loops in Java

10. A good model is one that you can make assertions about. What else is important about assertions?

A. Only bullies make assertions

B. If assertions cannot be coded directly in your programming language, write automated unit tests for them

C. Assertions encourage the situation where the only way to understand a program is to trace execution through branching paths

D. Assertions define contracts of services and entity modifiers

E. Avoid expressing assertions in the ubiquitous language

F. Write assertions into documentation or diagrams where it fits the style of the project's development process

1. Which of the following is the definition of Ubiquitous Language given in class?

A. A handful of classes and patterns that shape the core flow of the system being built
B. A language structured around the domain model and used by all team members within a bounded context to connect all the activities of the team with the software
C. The logical architecture of the system
D. An enterprise data model, forged by Sauron in the fires of Mordor

2. Which of the following are true about Ubiquitous Language?

A. The team should work toward a language that enables natural, precise statements about the model and domain
B. Increases the amount of translation for team members. Translation is good!
C. Cultivates a language of domain terms, not tech terms
D. All team members should exercise the ubiquitous language - a lot
E. Increases the distance between domain experts, developers, requirements and code
F. A change in the ubiquitous language is a change in the model
G. Experimentation with language is critical to developing a truly ubiquitous language
H. Provides a higher-level language to describe and discuss service contracts in speech or writing

1. Which of the following statements are true about entities?

A. Have a thread of identity over time
B. Are immutable
C. Focus their definition on identity and lifecycle
D. Fill them up with as many properties as possible
E. Exist in the real world
F. Cannot be Aggregate roots
G. We should try to model most concepts as entities

2. Which of the following statements are true about value objects?

A. Exist in the real world (eg. a dollar bill is always going to be a value object)
B. Have a thread of identity over time
C. Are immutable
D. Are equal when all their properties are equal
E. Cannot have behavior
F. We should be biased towards modeling concepts as value objects
G. Can be aggregate roots, or whatever they want to be. They are in charge of their own destiny and nothing can stand in their way!
H. Must have side-effect-free functions
I. Improve encapsulation

3. Aggregates help us deal with which of the following types of boundaries?

A. Transactional
B. International
C. Distribution
D. Nation-state
E. Concurrency
F. Ubiquitous language

4. What are the guidelines for first-pass aggregate boundaries?

A. Cross your fingers and hope for the best
B. Look for objects that together form a "conceptual whole"
C. Don't bother defining aggregate boundaries, they are over-rated
D. The "delete" rule of thumb
E. Get aggregate boundaries correct the first time, because they can *never* change

5. Which are the following are true statements about aggregates?

A. Data consistency rules apply within aggregates, so each aggregate is its own "island" of data consistency
B. Aggregates can be embedded within each other
C. Within an aggregate, invariants apply at every transaction commit
D. The aggregate root must be a value object
E. The aggregate root must be an entity
F. Aggregates enable data consistency rules to be represented within the domain model
G. Aggregates are a waste of time, because everything should be connected to everything else. Barrel of Monkeys! Jenga!

6. When making changes across aggregate boundaries, what should we keep in mind?

A. Aggregates are "eventually consistent" with each other
B. Aggregates cannot ever be shared between bounded contexts, except in a shared kernel
C. All aggregates must always be kept consistent with each other
D. Aggregate-related updates propagate asynchronously through the system
E. Domain events are the only correct way to handle cross-aggregate updates
F. We are ok with data being stale between aggregates, at least for a period of time

7. What is true about the relationship between repositories and aggregates?

A. Query access to aggregates is provided via repositories (which provide the illusion of an in-memory collection of all objects of that aggregate's root type)
B. Aggregates can only be persisted through repositories that save to relational databases using an object-relational mapper
C. *What* relationship? Aggregates and repositories have nothing to do with each other.
D. Repositories must inherit from a base class that provides CRUD (Create-Read-Update-Delete) functionality
E. Repositories must always be suffixed with the word "Repository" (for example, CargoRepository)
F. Repositories are used to create aggregates for the very first time (i.e. when aggregates do not yet exist in the system)

8. Domain events are defined as something that happened that domain experts care about. What else is true about domain events?

A. Provide a good mechanism for synchronizing across aggregate boundaries
B. Get in the way of creating clearer, more expressive models
C. Model chronology (time) explicitly within a domain model
D. Are useful for enabling high performance systems
E. Cannot trigger other domain events
F. Are a full-fledged part of the domain model
G. Allow subsystems to be decoupled using event streams
H. Cannot be used to represent the state of entities

9. Which of the following statements are true about services in DDD?

A. Domain services are always used to orchestrate entities and value objects
B. Service-Oriented Architecture (SOA) and DDD are complementary
C. Application services and domain services are essentially the same thing
D. Domain services have an identity and lifecycle
E. Domain services are a great way to perform a significant business process, or calculate a Value requiring input from more than one domain object
F. Service contracts should be kept as flexible as possible by deliberately keeping them vague

1. What are potential benefits of context mapping?

A. Mapping the future configuration allows you to create an enterprise modeling strategy
B. Context maps are a big-picture complement to core domain
C. The process of mapping contexts makes the state of context boundaries more clear
D. The resulting context map will help you see where context boundaries are not aligned with subsystem boundaries
E. **Context mapping is a helpful technique for non-technical people to see the technical and organizational landscape**

2. What is true about context maps?

A. Map the future!
B. They need to be right first-time
C. An accurate context map requires knowledge of what is happening in development
D. We only care about our perspective
E. Ignore messy realities and make them as idealistic as possible
F. Context Map reflects what is. It is still left to us to make implementation decisions that make the best of it.

3. Why is it important to embrace the reality that there are always multiple models of the domain?

A. Each model can be targeted towards solving certain specific business problems most effectively
B. We want more models so we can write more software
C. It reduces overall complexity
D. We don't – there should only ever be one, true, model of the domain

4. Which of the following are accurate definitions of a bounded context?

A. A DBContext in Entity Framework
B. The conditions under which a particular model is defined and applicable
C. A subsystem boundary or business component
D. A Bounded Context is linguistic: a part of the system/project where language is consistent and rules agree
E. A description of a boundary (typically a subsystem, or the work of a particular team) within which a particular model is defined and applicable
F. The setting in which a word or statement appears that determines its meaning

5. What are practices to follow within a bounded context?

A. Explicitly define the context within which a model applies
B. Have multiple models interacting within the bounded context
C. Explicitly set boundaries in terms of team organization, usage within specific parts of the application, and physical manifestations such as code bases and database schemas
D. Use the same language in diagrams, writing, and especially speech
E. Have multiple ubiquitous languages within the bounded context
F. Apply Continuous Integration to keep model concepts and terms strictly consistent within these bounds
G. Use multiple development processes within the context
H. Exercise the ubiquitous language to hammer out a shared view of the model

6. Classify each of the following practices according to which single subdomain they might *most* be appropriate

A. Outsource development to an offshore team [Core | Generic | **Supporting**]
B. Buy an off-the-shelf software package [Core | **Generic** | Supporting]
C. Do model exploration [**Core** | Generic | Supporting]
D. Do lots of experiments and make many mistakes [**Core** | Generic | Supporting]
E. *Never* buy an off-the-shelf software package [**Core** | Generic | Supporting]
F. Use an open source framework [Core | **Generic** | Supporting]
G. Collaborate intensively with the domain experts [**Core** | Generic | Supporting]
H. Bring in a team of contractors [Core | Generic | **Supporting**]
I. Hire an intern to do the work [Core | Generic | **Supporting**]
J. Isolate in an explicitly bounded context [**Core** | Generic | Supporting]

7. Which of the following should you do when you are in partnership with another team?

A. Not too much, just have the team-leads meet for lunch every Friday
B. Institute a process for coordinated planning of development and joint management of integration
C. Build an anti-corruption layer between the two contexts
D. Cooperate on the evolution of your interfaces with the other team to accommodate the development needs of both systems
E. Schedule interdependent features so that they are completed for the same release

8. What is necessary to support a shared kernel?

A. Designate with an explicit boundary some subset of the domain model that the teams agree to share
B. Keep the kernel small
C. Treat the explicitly shared stuff as having special status
D. Change the shared stuff whenever you want, without consultation with the other team
E. Define a Continuous Integration process that will keep the kernel model tight and align the ubiquitous language of the teams

9. If you are downstream from a team and have adopted their model because it is good enough for your purposes, which strategic pattern are you most likely to be following?

A. Customer/supplier
B. Anti-corruption layer
C. Partnership
D. Conformist
E. Shared kernel

10. How is a translation layer different for cooperative partner teams versus an anticorruption layer?

A. Cooperative partner teams don't need automated tests
B. Anticorruption layers must always be maintained by both teams
C. Translation is usually easier for cooperative partner teams
D. Anticorruption layers only ever translate in one direction

11. What are some of the advantages of taking a Conformist approach to an upstream model?

A. It eliminates the complexity of translation between bounded contexts by slavishly adhering to the model of the upstream team
B. It cramps the style of the downstream designers
C. Conformity enormously simplifies integration
D. Share a ubiquitous language with your upstream team
E. The upstream is in the driver's seat, so it is good to make communication easy for them
F. The upstream team will likely never share information with you

12. An open host service defines a protocol that gives access to your subsystem as a set of services. What is also true about an open host service?

A. Write a different implementation of the service for each downstream context
B. Use a one-off translator to augment the protocol for any idiosyncratic, special case so that the shared protocol can stay simple and coherent
C. Open the protocol so that all who need to integrate with you can use it

13. When dealing with a sprawling system filled with contingent logic and lacking boundaries, what are appropriate steps to take?

A. Rewrite it whole, since this has been proven to be the quickest path to success
B. Investigate the use of an anticorruption layer to protect your new context from the existing sprawling system context if you need to do sophisticated modeling
C. Put a lot of effort into identifying its context boundaries
D. Do not try to apply sophisticated modeling within this context
E. Be alert to the tendency for such systems to sprawl into other contexts
F. Draw a boundary around the entire mess and label it a "Big Ball of Mud"

14. Which of the following are true about upstream-downstream relationships?

A. The "upstream" group's actions affect project success of the "downstream" group
B. Data must flow from "upstream" to "downstream"
C. The actions of the downstream do not significantly affect projects upstream
D. The downstream team may succeed independently of the fate of the upstream team

15. If you are downstream from a sprawling legacy system and need to innovate in your context because that is where the core domain is, which strategic pattern are you most likely to adopt?

A. Customer/supplier
B. Anti-corruption layer
C. Partnership
D. Conformist
E. Shared kernel

16. Which of the following statements about core domain are true?

A. Core domain is primarily about core features
B. Market differentiation is a key component to focus on when trying to identify your core domain
C. There is always only one core domain at any one time
D. Core domain is primarily about core business
E. Spend the effort in the core to find a deep model and develop a supple design
F. The core domain never changes
G. Keep the core domain small
H. Apply top talent to the core domain, and recruit accordingly
I. Justify investment in any other part by how it supports the distilled core

17. What are the guiding principles of DDD?

A. Focus on the core domain (i.e. complexity and business opportunity are high)
B. Entities are evil
C. Explore models in a creative collaboration of domain practitioners and software practitioners
D. CQRS is the only true architectural pattern for implementing DDD correctly
E. Collaboration with the business is a waste of time
F. Speak a ubiquitous language within an explicitly bounded context
G. Always use repositories, or face the summative wrath of Eric Evans!